UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISTRIBUCIÓN MULTICAST DE VÍDEO EN DIRECTO SOBRE LA RED MÓVIL LTE

Alumno: Ignacio Domínguez Martínez-Casanueva

Tutor: Luis Bellido Triana

MADRID 2015

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Distribución multicast de vídeo en directo sobre la red móvil LTE

Autor: D. Ignacio Domínguez Martínez-Casanueva

Tutor: D. Luis Bellido Triana

Departamento: Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL

Presidente: Dña. Encarnación Pastor Martín

Vocal: D. Gabriel Huecas Fernández-Toribio

Secretario: D. Luis Bellido Triana

Suplente: D. David Fernández Cambronero

RESUMEN

En la última decada, la telefonía móvil ha evolucionado a una extraordinaria velocidad, permitiéndonos acceder a funcionalidades características de los PC pero con la ventaja de poseer una movilidad total. Con la aparición de la tecnología Long Term Evolution (LTE), comunmente conocida como 4G, se ha conseguido desarrollar un sistema que se ha mejorado notablemente las prestaciones proporcionando alta velocidad y eficiencia a los ya masivamente utilizados smartphones. Gracias a este exponencial incremento del ancho de banda disponible, los usuarios hoy en día no se conforman sólo con navegar por páginas Web, sino que cada vez muestran un mayor interés en poder explotar al máximo los recursos multimedia, dando lugar a servicios como el streaming de vídeo.

De este modo, a raíz del proyecto LTExtreme centrado en el análisis y la propuesta de optimización para servicios de streaming multimedia multicast/unicast sobre la tecnología LTE, surge este trabajo en el cual se pretende extender dicho análisis a la multidifusión de vídeo en directo. El proyecto se basa en la implementación de la arquitectura propuesta por el organismo 3GPP para dar este servicio, considerándose como una solución eficiente en la que se combina el protocolo de transporte multicast FLUTE (File Delivery over Unidirectional Transport) con la tecnología DASH (Dynamic Adaptative Streaming over HTTP).

La arquitectura se ha implementado mediante la creación y configuración de una maqueta de laboratorio gracias a la herramienta de virtualización Virtual Networks over linuX (VNX). Un escenario simplificado de la red móvil LTE junto con el servidor de contenidos y varios clientes móviles, pudiendo realizar simulaciones de una emisión de vídeo en directo, y a su vez analizar los resultados obtenidos, así como la calidad de servicio percibida. Concretamente, se realizará un análisis de los problemas asociados a los casos de uso tratados, tanto de la emisión de un único vídeo como una de duración infinita, asemejándose a lo que supondría la emisión de la programación televisiva para un determinado canal. Por último, se plantearán ideas surgidas a raíz de los resultados obtenidos de dichos estudios y que puedan tener futuro y ser aplicables al mundo real.

PALABRAS CLAVE

Streaming, vídeo en directo, smartphone, DASH, FLUTE, LTE, virtualización, multicast, IGMP, FEC, segmento de vídeo, contenido multimedia, servidor de contenidos.

SUMMARY

Mobile phone capabilities have dramatically evolved in the last decade, allowing users to access typical PC features with a high mobility advantage. With the Long Term Evolution (LTE) technology appearance, more commonly known as 4G, it has been developed a new system which greatly leverages perfomance providing widely used smartphones an efficient and high speed connection to network. Thanks to this exponential growth of bandwidth availability, nowadays users can´t just live with surfing the Web since they show more and more interest in enjoying media resources such as video streaming services.

Therefore, project LTExtreme focused on analyzing and optimizing media multicast/unicast streaming services over LTE technology, was the starting point for this work whose target is live video streaming. This project is based on an implementation of the architecture selected by the 3GPP organization in order to provide this sort of service. This particular architecture is believed to be an efficient solution that combines multicast transport protocol FLUTE (File Delivery over Unidirectional Transport) with DASH (Dynamic Adaptative Streaming over HTTP) technology.

3GPP selected standard has been implemented by creating and configuring a lab testbench thanks to the virtualizing tool called Virtual Networks over linuX (VNX). The result is simplified scenario of the LTE network alongside media content server and multiple clients, so we could simulate a live video streaming as well as analize obtained results such as perceived quality of service. In particular, there will be a study of problems related to both analized cases of use: single video transmission and infinite duration video transmission, whose goal is emulating a typical TV channel streaming. Finally, as a result of these studies multiple ideas will be discussed due to their potential application in the near future.

KEYWORDS

Streaming, live video, smartphone, DASH, FLUTE, LTE, virtualization, multicast, IGMP, FEC, media content, video segment, content server.

GLOSARIO

3GPP	3rd Generation Partnership Project
CBQ	Class-based queuing
CDN	Content Delivery Network
DASH	Dynamic Adaptative Streaming over HTTP
eMBMS	Evolved Multimedia Broadcast and Multicast Service
EPC	Evolved Packet Core
EPS	Evolved Packet System
EUTRAN	Evolved UTRAN
FEC	Forward Error Correction
FDT	File Delivery Table
FLUTE	File Delivery over Unidirectional Transport
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IGMP	Internet Group Management Protocol
KVM	Kernel-based Virtual Machine
LTE	Long Term Evolution
LXC	Linux Containers
MPD	Media Presentation Description
MSE	Media Source Extension
OFDM	Orthogonal Frequency-Division Multiplexing
QCOW	QEMU Copy On Write
TC	Linux Traffic Control
TOI	Transport Object Identifier
TSI	Transport Session Identier
URI	Uniform Resource Identifier
VM	Virtual Machine
VNX	Virtual Networks over linuX
XML	eXtensible Markup Language

ÍNDICE

1. INTRODUCCIÓN

1.1 Introducción y motivación

En la última decada, la telefonía móvil ha evolucionado a una extraordinaria velocidad, permitiéndonos acceder a funcionalidades características de los PC pero con la ventaja de poseer una movilidad total. Gracias a la aparición de la tecnología 3G, se presentó a los usuarios la posibilidad real de poder hacer uso de Internet a través de sus terminales, pero ha sido la cuarta generación, Long Term Evolution (LTE) y Long Term Evolution Advanced (LTE-Advanced), la que junto con la popularización de los smartphones y módems USB, ha provocado una explosión en el sector de las telecomunicaciones. La tecnología 4G ha conseguido desarrollar un sistema en el que se ha mejorado notablemente la capacidad, la eficiciencia y las prestaciones llegando a alcanzar caudales de 100 Mbps\50 Mbps y una latencia inferior a 10 milisegundos [1]. Este éxito es debido en parte al uso de un robusto interfaz de radio basado en Orthogonal Frequency-Division Multiplexing (OFDM) al igual que sistemas como WiFi o WiMAX, pero en especial ha sido gracias a la transición a la conmutación de paquetes. El abandono de la conmutación de circuitos para trasladarse completamente al modelo All-IP ha dado lugar a una arquitectura plana, con pocos elementos, que proporciona alta velocidad y eficiencia dada su perfecta integración con otras redes de datos IP, como pueden ser redes privadas o la propia Internet. Es por esta similitud con otras redes de datos por la que es posible simular un escenario real en un entorno virtual con el objetivo de investigar las nuevas funcionalidades que nos pueda aportar la tecnología 4G.

Gracias al exponencial incremento del ancho de banda disponible en nuestros smartphones ya no nos conformamos sólo con navegar por páginas Web, sino que cada vez nos interesa más el poder explotar al máximo los recursos multimedia. Especialmente, están suscitando un gran interés los streamings de audio como pueden ser Spotify o Apple Music, y por otro lado la visualización de vídeo, principalmente bajo demanda, como puede ser el caso de Youtube o Netflix. En nuestro caso va a ser objeto de estudio el servicio de vídeo, concretamente la emisión en directo. No obstante, estamos hablando de telefonía móvil, por lo que en comparación con los PC existe el inconveniente que supone el acceso mediante un enlace radio. Este hecho da lugar a variaciones como pueden ser del ancho de banda o de la tasa de errores según la posición del usuario con respecto al eNodeB. Así pues, apoyándonos en una serie de protocolos para mitigar dichas amenazas, vamos a analizar el rendimiento obtenido en una emisión de vídeo de este tipo sobre un escenario LTE simulado con el fin de detectar si proporcionar este servicio es viable y con un alto grado en prestaciones.

1.2 Objetivos y metodología

Partimos de la base establecida por el proyecto INNPACTO llamado LTExtreme [2], compuesto por un consorcio entre Alcatel-Lucent, la Universidad Carlos III de Madrid y la Universidad Politécnica de Madrid, el cual se trata de un proyecto centrado en el análisis y la propuesta de optimización para servicios de streaming multimedia multicast/unicast sobre la tecnología LTE. Así pues, surgiendo como derivación del punto de partida que se acaba de identificar, concretamente en este trabajo de investigación se plantea como objetivo principal conseguir simular la transmisión de vídeo de un evento en directo, como podría ser un partido de fútbol, a un grupo de usuarios con alta calidad. Nos apoyaremos en el uso de la tecnología DASH (Dynamic Adaptative Streaming over HTTP) con el fin de poder emitir con la mejor calidad de vídeo posible para cada usuario, puesto que el escenario se trata de una red móvil LTE y los terminales accederán al streaming sobre enlaces con distintas características (latencia, ancho de banda, etc). Por otro lado, coordinando el trabajo se encuentra el protocolo FLUTE (File Delivery over Unidirectional Transport), cuya función es el transporte multicast de los segmentos multimedia a los smartphones de los clientes.

Con respecto a la metodología, el trabajo partirá de la configuración de una maqueta basada en el uso de máquinas virtuales gracias al software de virtualización Virtual Networks over linuX (VNX) [3], desarrollado en el Departamento de Ingeniería Telemática (DIT) de la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT). Mediante esta herramienta de virtualización que utiliza contenedores ligeros Linux Containers (LXC), se emulará una red móvil al completo pudiendo realizar y analizar medidas de la calidad del servicio que se está dando. Con el objetivo de simplificar dicho escenario, se hará uso únicamente de cuatro Virtual Machines (VMs): Servidor, Red, Cliente 1 y Cliente 2. En cuanto a la transmisión de los segmentos en este escenario, se llevará a cabo mediante un programa escrito en lenguaje C++ que permite implementar FLUTE, para que a través de un envío multicast los usuarios que se unan el grupo puedan recibir los datos multimedia. Por otro lado, la herramienta utilizada para la implementación del conjunto de tecnologías DASH, se trata de un cliente JavaScript completo, ejecutándose en los clientes, el cual proporciona un sencillo reproductor, de modo que los usuarios puedan disfrutar del servicio en su totalidad vía Web.

Así pues, previamente se llevará a cabo un estudio a fondo sobre el uso de estas herramientas en cuestión, al igual que de todas aquellas complementarias que sean necesarias en el desarrollo del proyecto, como puede ser la realización de Shell scripts. No obstante, el estudio no se limitará a las herramientas utilizadas, si no que también serán objeto de ello elementos conceptuales clave para el funcionamiento del servicio, como son la transmisión de datos en modo multicast, la implementación de FLUTE y DASH, o los mecanismos seguidos para la reproducción multimedia. Además, se realizará un análisis de los problemas asociados a los casos de uso tratados, tanto de la emisión de un único vídeo como una de duración infinita. Por último, se plantearán ideas surgidas a raíz de los resultados obtenidos de dichos estudios y que puedan tener futuro y ser aplicables al mundo real.

2. ESTÁNDAR MPEG-DASH

2.1 Streaming de vídeo sobre LTE

La enorme creciente demanda de servicios de streaming de vídeo en cualquier lugar y en cualquier momento supone enfrentarse a una serie de desafíos. Las redes Long Term Evolution (LTE) presentan obstáculos como puede ser el ancho de banda disponible o la calidad del canal radio. Para ello el 3rd Generation Partnership Project (3GPP) propone el uso de Dynamic Adaptative Streaming over HTTP (DASH) [4] como el estándar a seguir para los servicios de streaming multimedia en LTE. DASH consiste en un conjunto de tecnologías que trabajan sobre robusto protocolo HTTP, permitiendo al cliente en un streaming de vídeo adaptarse a las condiciones de la red eligiendo la calidad de vídeo que se adecue más. Por lo tanto, DASH es de gran utilidad para un usuario que desea acceder a servicios de video bajo demanda (VoD), ya que la provisión del servicio a cada usuario es personalizada, al enviar calidades de vídeo distintas según sus características gracias a HTTP.

Sin embargo, el 3GPP también define cómo DASH puede utilizarse para ofrecer difusión de vídeo sobre LTE. Para ello se hace uso del servicio estándar de datos para multicast/broadcast definido por el 3GPP, el llamado Evolved Multimedia Broadcast and Multicast Service (eMBMS). Dicho servicio de datos hace uso de un canal multicast común para enviar la misma información a un conjunto de clientes. Este mecanismo mejora sustancialmente la eficiencia en el uso de los recursos de la red, puesto que en una difusión sobre LTE los segmentos de vídeo DASH enviados a la red multicast serán recibidos por igual en todos los miembros del grupo multicast, utilizando así menor cantidad de espectro a la par que pudiendo compartir los recursos con otras sesiones unicast. Es por este motivo por el cual no son necesarias distintas representaciones del vídeo, ya que el servidor enviará una única calidad de vídeo por el canal multicast, de modo que a todos los usuarios les lleguen los mismos datos. Así pues, gracias al canal eMBMS se mejora sustancialmente la utilización de las frecuencias en las celdas LTE, un recurso que es bastante limitado.

2.2 Modelo híbrido del 3GPP

Como hemos visto en el capítulo anterior, el 3GPP define como DASH también puede ser utilizado para proporcionar servicios de broadcast de vídeo en LTE mediante el canal eMBMS. Dicho objetivo se materializa sustituyendo las transmisión unicast de segmentos de vídeo con HTTP por una transmisión de tipo multicast. Por otro lado, este tipo de transmisión supone la existencia de una cierta tasa de error, por lo que técnicas Application Layer - Forward Error Correction (AL-FEC) [5] son utilizadas para la recuperación de paquetes perdidos. Así pues, para el envío de los ficheros el 3GPP propone el protocolo File Delivery over Unidirectional Transport (FLUTE) [6], capaz de encapsular los segmentos de video con la codificación del estándar DASH y con la redundancia de las técnicas AL-FEC.

No obstante, el uso de técnicas de correción FEC no garantiza que los segmentos enviados mediante FLUTE por el canal eMBMS sean recuperados por el cliente si se han recibido con errores. Se puede dar la situación en la cual la tasa de error sea tan grande que se pierdan un número de paquetes suficiente como para no poder recuperar el segmento de vídeo. Esta situación puede darse incluso con códigos altamente eficientes como pueden ser los códigos Raptor. Para ello el 3GPP introduce como posible solución el uso de una arquitectura híbrida [24] en la que se combinan FLUTE y DASH. Dicha arquitectura posee la capacidad de recuperar aquellos segmentos de vídeo que se han perdido mediante retransmisiones unicast sobre HTTP. En la figura 2.1 se pueden observar todos los mecanismos seguidos para la coordinación entre DASH y FLUTE.

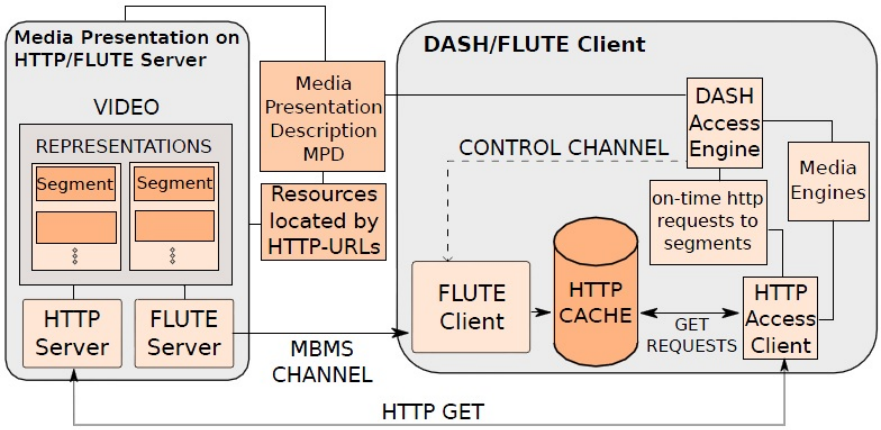

Figura 2.1 : Modelo híbrido FLUTE/DASH

Los segmentos de vídeo son transmitidos con FLUTE sobre una sesión multicast entre el servidor y los usuarios asociados al grupo. El cliente FLUTE en el terminal recibe los paquetes, decodifica el segmento de vídeo utilizando las técnicas AL-FEC, y guarda el segmento en una caché HTTP. Mediante el uso del HTTP Access Client, el motor DASH Access Engine puede obtener el segmento de vídeo de la caché cuando el segmento se ha recibido correctamente bien a través de la sesión multicast de FLUTE, o bien, de una transmisión unicast previa. En cambio, si el segmento de vídeo no se encuentra en la caché cuando el DASH Access Engine lo solicita, este es recuperado mediante una retransmisión unicast por el HTTP Access Client y almacenado en la propia caché. Por tanto, de acuerdo a esta configuración el envío de datos mediante FLUTE y el modelo multicast son transparentes al motor DASH Access Engine, mientras que la recuperación de segmentos perdidos utilizando HTTP unicast es automaticamente llevada a cabo por el HTTP Access Client cuando el DASH Access Engine lo solicita pero no lo encuentra en la caché HTTP.

2.3 File Delivery over Unidirectional Transport (FLUTE)

FLUTE se trata de un protocolo cuya función es la transmisión unidireccional de ficheros tanto de pequeño como de gran tamaño. Es decir, requiere una conexión entre el servidor y el cliente mientras que no es necesaria una entre el cliente y servidor. Además, el diseño de FLUTE, que trabaja sobre UDP, permite realizar envíos unicast pero su mayor cualidad es el envío multicast. En otras palabras, tiene la potente capacidad de distribuir datos a una gran cantidad de clientes simultáneamente. Esta característica se debe a que el principal objetivo de este protocolo de transporte es conseguir una enorme escalabilidad. Dicho esto, junto con su capacidad para trabajar en cualquier tipo de red como LANs, WANs, Inalámbricas o la propia Internet, nos da como resultado un protocolo totalmente adecuado para la distribución de ficheros en escenarios multidifusivos. Para la distribución de ficheros FLUTE se fundamenta en dos elementos que detallaremos a continuación, uno es la File Delivery Session y el segundo se conoce como File Delivery Table (FDT).

2.3.1 File Delivery Session

El protocolo de distribución de contenidos FLUTE se considera una extensión de los protocolos de transporte ALC/LCT [7] [8], de los cuales se reutiliza la idea de la sesión pero que FLUTE la adopta con el nombre de File Delivery Session. Esta sesión es un conjunto de canales ALC/LCT lógicamente agrupados, donde cada uno de ellos es una asociación entre el servidor y una determinada dirección IP asociada al canal de transmisión. Así pues, es tan simple como que el cliente se una a dicho canal para recibir los datos de la difusión, o para abandonarlo cuando lo considere oportuno.

No obstante, hay una serie de potenciales problemas que surgen a raíz de esta capacidad, como por ejemplo, un servidor puede estar realizando en paralelo múltiples sesiones con distintos contenidos y distintos destinatarios. Así pues, ¿Cómo podemos diferenciar una sesión de otra? Para ello los paquetes de datos que son enviados a través de los canales que hemos mencionado llevan sus correspondientes cabeceras ALC/LCT con un campo en especial llamado Transport Session Identier (TSI). De este modo, el cliente al recibir un fichero va a ser capaz de identificar unívocamente una sesión con el par de valores dirección IP + TSI. Sin embargo, con la adopción de esta medida no solventamos todos los problemas de este tipo puesto que en una sesión se pueden enviar una enorme cantidad de ficheros, por lo que también necesitamos poder diferenciar cada objeto recibido en una sesión determinada. Pues bien, de una forma similar a la sesión, los paquetes de datos en sus cabeceras ALC/LCT también van a incluir un campo llamado Transport Object Identifier (TOI), el cual a su vez permite identificar a cada objeto dentro de una File Delivery Session.

2.3.2 File Delivery Table (FDT)

Con la finalidad de informar al cliente de los objetos que van a ser recibidos en la sesión FLUTE, se le envía el llamado File Delivery Table, o también FDT. Este fichero se trata de un archivo con metadatos en eXtensible Markup Language (XML), en el cual se describen las propiedades de los objetos transmitidos dentro de una File Delivery Session. Entre las características del FDT podemos destacar que en la descripción de que cada objeto se debe indicar su TOI correspondiente, el Uniform Resource Identifier (URI) que identifica el fichero y el tamaño exacto que posee. Esto como se ha explicado anteriormente es necesario, puesto que un objeto no descrito con exactitud en el FDT de una sesión será descartado. También puede incluir información relativa a la codificación FEC en el caso de que sea utilizada, parámetros que nos serán de utilidad en capítulos posteriores donde se pondrá a prueba esta capacidad que nos ofrece FLUTE.

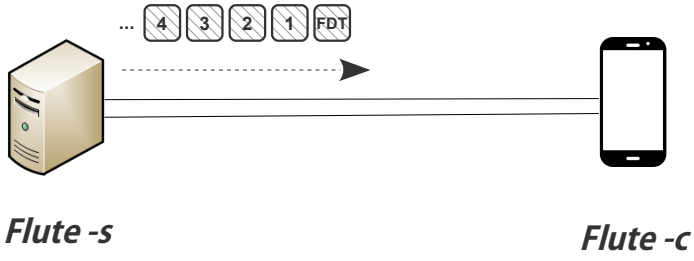

Figura 2.2 : Envío FDT + sucesivos TOIs

El mecanismo que se sigue en el cliente se basa en la construcción de una base de datos de FDT. Esto se debe a que dentro de una sesión el FDT se envía en forma de instancias FDT, lo que va a permitir ir actualizando la base de datos con las descripciones de los objetos a recibir a medida que transcurre la sesión (Figura 2.2). Con el fin de detectar estas actualizaciones en el transcurso cada instancia en sus cabeceras lleva un campo llamado FDT Instance ID que permite al receptor identificarla. A raíz de esta caracterisitica surge otra medida que se debe tomar obligatoriamente, una medida que consiste en que el valor del identificador TOI, que hemos presentado anteriormente, para las instancias FDT debe ser siempre cero. Por lo tanto, todos los objetos que son enviados en una sesión deben llevar un TOI con un valor mayor que cero, puesto que el valor TOI=0 está reservado para el envío de instancias FDT.

2.4 Dynamic Adaptive Streaming over HTTP (DASH)

El conjunto de tecnologías DASH, recomendado por el 3GPP para proporcionar un servicio de streaming multimedia sobre LTE, tiene la especial característica de permitir al cliente adaptarse a las condiciones de la red. Así pues, el cliente que recibe un streaming de vídeo. está capacitado para seleccionar entre las distintas representaciones del mismo contenido multimedia codificado en diferentes calidades. De esta forma, en un servicio de estas características se le concede el control principalmente al cliente. El mecanismo utilizado

por DASH se basa en la descripción del contenido que se está emitiendo en múltiples elementos con aportaciones de información distinta como puede ser la duración o la codificación del vídeo. La organización de toda esta información sigue una estructura basada en un modelo jerárquico que explicaremos a continuación.

2.4.1 Orden jerárquico

La colección de las distintas versiones codificadas y entregables del contenido multimedia y su correspondiente descripción forma lo que se conoce como Media Presentation. DASH se basa en un modelo de datos que sigue un orden jerárquico (Figura 2.3) cuyo primer elemento describe la presentación multimedia y recibe el nombre de Media Presentantion Description (MPD).

El contenido multimedia del streaming puede estar formado por uno o varios Periods en el tiempo. Dentro de un Period el material está organizado en los llamados Adaptation Sets, los cuales agrupan todas las versiones codificadas para cada componente multimedia. Típicamente estos componentes son el vídeo principal y el audio principal, no obstante, podríamos tener distintas señales de vídeo y de audio en un mismo Period. Por ejemplo, en una carrera de motos podemos recibir al mismo tiempo las señales procedentes de distintas cámaras como puede ser una aérea y otra en la parte trasera de la moto, mientras que para el audio podemos escuchar una señal de comentaristas hablando en español y otra en inglés. A su vez en el interior de cada grupo Adaptation Set podemos encontrar distintas versiones codificadas que reciben el nombre de Representation. Un ejemplo puede ser que para el Adaptation Set del vídeo principal tengamos varias versiones de ese fragmento de vídeo en distintas calidades 480, 720 y 1080p.

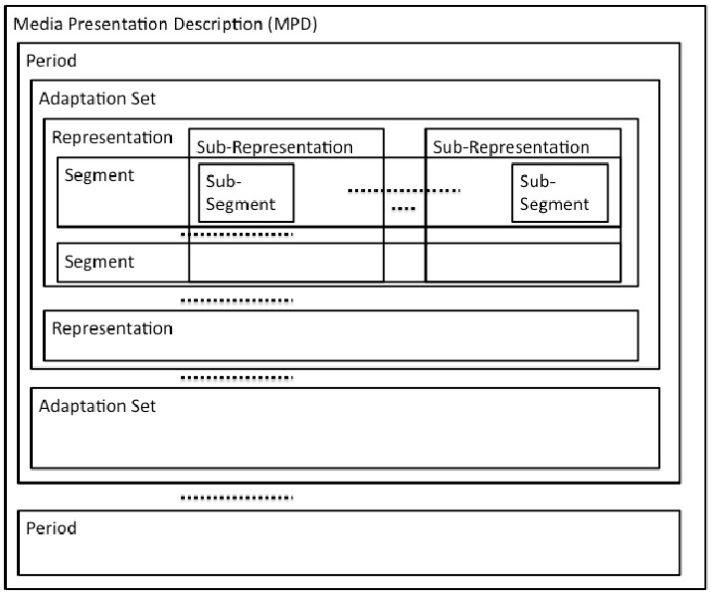

Figura 2.3 : Modelo de datos de alto-nivel en DASH

Si seguimos descendiendo en la jerarquía podemos apreciar que cada Adaptation Set puede contener una gran cantidad de segmentos. El Segment es una división en el tiempo del conjunto de datos listo para el acceso y la entrega. Típicamente todos los segmentos tienen la misma duración, así que para hacernos una idea un periodo de 20 segundos podría dividirse en 10 segmentos que tienen 2 segundos de duración. Cada uno de estos segmentos será solicitado y visualizado por el usuario hasta completar los 20 segundos que dura el fragmento de video. Podemos ir más allá e incluso subdividir los Segments y las Representations en Sub-Segments y Sub-Representations respectivamente, pero para nuestro caso de estudio sólo son mencionados puesto que no van a ser necesarios.

2.4.2 Media Presentation Description (MPD)

DASH identifica los distintos segmentos de una representación de vídeo por su Uniform Resource Locator (URL), y la definición de las distintas representaciones y segmentos se encuentran almacenadas en un fichero Media Presentation Description (MPD). Dicho fichero es descargado por el cliente del streaming con el objetivo de saber cómo se ha preparado el contenido multimedia y como puede ser accedido. Valiéndose de la información proporcionada por el MPD, el cliente del streaming aplicará un algoritmo de adaptación para reproducir el vídeo seleccionando los segmentos con el bitrate más adecuado al ancho de banda disponible.

El fichero MPD es concretamente un documento en formato XML, por lo que vamos a poder definir fácilmente la descripción del streaming en cuestión. Este diseño se lleva cabo referenciando a cada elemento de la jerarquía mediante el uso de las etiquetas características de XML, como por ejemplo un Adapation Set o un Segment. La definición de cada uno de estos elementos con sus principales atributos será detallada en el capítulo 4.3.1, puesto que nos servirá de gran ayuda para entender el posterior desarrollo de los experimentos en directo que se han realizado en este trabajo.

3. HERRAMIENTAS DISPONIBLES

3.1 Virtual Networks over linuX (VNX)

Para la creación y configuración de la maqueta que simula un escenario LTE se ha utilizado la herramienta de virtualización Virtual Networks over linuX (VNX) . Desarrollada en el Departamento de Ingeniería Telemática (DIT) de la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT), VNX es una herramienta open-source que nos permite desplegar rápidamente redes y nodos virtuales sobre los que poder probar nuestras aplicaciones y servicios con facilidad, en nuestro caso el servicio de streaming de vídeo en directo.

En el despliegue del entorno virtual, VNX parte de la generación de un sistema de ficheros, conocido como "root filesystem", el cual es compartido por todos los nodos virtuales inicialmente. Para poder crear este sistema de ficheros común a todos se hace uso del formato de disco QCOW (QEMU Copy On Write) el cual requiere muy poco espacio y es capaz de expandirse dinámicamente. Este proceso se fundamenta principalmente en el uso de contenedores Linux Containers (LXC), aunque también soporta el hipervisor Kernel-based Virtual Machine (KVM) de modo que podríamos desplegar máquinas virtuales con sistemas operativos tales como Windows 7 o CentOS. No obstante, para nuestro caso de estudio utilizaremos los Linux Containers basados en Ubuntu dada la libertad que nos ofrece y los pocos recursos que consumen. Esta es una importante característica de los contenedores debida a que todos comparten el mismo kernel del sistema operativo pero al mismo tiempo se encuentran perfectamente aislados los unos de los otros mediantes "namespaces". Por tanto, gracias a esta separación de procesos los nodos virtuales parten del sistema de ficheros inicial pero desde la propia inicialización del escenario de red cada uno va a poder tener una configuración totalmente distinta durante la ejecución virtual.

Otra atractiva característica de la orquestación que ofrece la herramienta VNX es el sencillo despliegue de la arquitectura de red. Para ello únicamente se necesita definir un fichero de formato XML que es leído por la librería libvirt, la cual tiene la función de gestionar tecnologías de virtualización tales como LXC, KVM, Xen, VMWare ESX o UML. Dicha librería, tras obtener todos los parámetros especificados en el XML, se va a coordinar con el hipervisor correspondiente para diseñar la red y configurar todos los nodos virtuales a nuestro gusto a partir del sistema de ficheros generado que le indiquemos a cada uno. Este último punto se debe matizar ya que a pesar de permitir a múltiples nodos utilizar un mismo sistema de ficheros VNX nos da la libertad de asignar en un mismo escenario distintos root filesystems según nos interese. Sin embargo, para nuestra maqueta utilizaremos sólo uno para todos los nodos dado que facilita su gestión y no requerimos un extremado nivel de independencia para cada nodo. Así pues, desplegar cualquier tipo de escenario es sumamente sencillo puesto que VNX sólo requiere cómo entrada un fichero XML correctamente configurado.

3.2 Video Tools

Muchos vídeos se encuentran pre-almacenados, y concretamente uno de ellos, Big Buck Bunny, es el que se ha utilizado para realizar los experimentos. No obstante, es posible añadir nuevos vídeos al servidor de contenidos. Los vídeos se han codificado en H.264 utilizando la herramienta x264 [9] a una resolución de 720p a diferentes bitrates. Mientras que para la creación de los segmentos de vídeo, se ha utilizado el programa MP4Box [10], que trocea el vídeo en segmentos de una determinada duración, e incluso es capaz de generar un fichero MPD para presentación multimedia.

3.3 Proyecto MAD Flute

El software que se ha decidido utilizar para la implementación del protocolo FLUTE surge de un proyecto nombrado MAD, el cual fue desarrollado en la Tampere University of Technology durante los años 2003 y 2004. La herramienta resultante de este proyecto, conocida como MAD-FLUTE [11], se trata de una herramienta de transferencia multicast de ficheros construida sobre una serie de librerías. Entre ellas destacamos la llamada MAD-ALCLIB que consiste en una implementación de los protocolos ALC/LCT junto con el protocolo de control de congestión RLC y esquemas de codificación FEC como Simple XOR o Reed-Solomon. Tal como se indica en el manual de usuario, accesible desde la web del proyecto, la aplicación nos ofrece una gran variedad de parámetros para configurar dependiendo de si está actuando como servidor o como cliente en la sesión FLUTE.

3.4 Squid

Tal como lo definen sus propios desarrolladores [12], Squid se trata de un proxy Web con caché que soporta protocolos como HTTP, HTTPS o FTP. Por lo tanto, esta herramienta nos va a aportar la caché en el dispositivo del cliente, que tal como hemos visto en el capítulo del estándar del 3GPP, es un elemento crucial en el funcionamiento del protocolo DASH para la visualización del vídeo. Además, accediendo a la terminal del Client podemos acceder a un log que nos proporciona el proxy con las peticiones realizadas al servidor web, de modo que esta cualidad nos va a ser sumamente útil para poder analizar el comportamiento en la reproducción de un vídeo.

3.5 Cliente Dash.js

Existen numerosas implementaciones del conjunto de tecnologías DASH como pueden ser bitdash [13] o la empleada por Microsoft Azure, pero para nuestro trabajo se ha optado por la herramienta llamada Dash.js. Esta implementación desarrollada por Dash-Industry-Forum [14] y Akamai, se trata de un cliente escritro en Javascript que incorpora un reproductor multimedia. Este cliente de DASH se trata de código "open source", por lo que si se desea tendremos total libertad para modificarlo y realizar nuestras propias variantes. Además cabe destacar que se trata de un proyecto en constante evolución, tal como se puede apreciar en su

repositorio de GitHub [15], del cual concretamente se ha lanzado como última release oficial la versión 1.4. Sin embargo, tiene como inconveniente el requerir un navegador que soporte Media Source Extension (MSE), por lo que por el momento es soportado en Google Chrome o Safari mientras que en Mozilla Firefox no es posible.

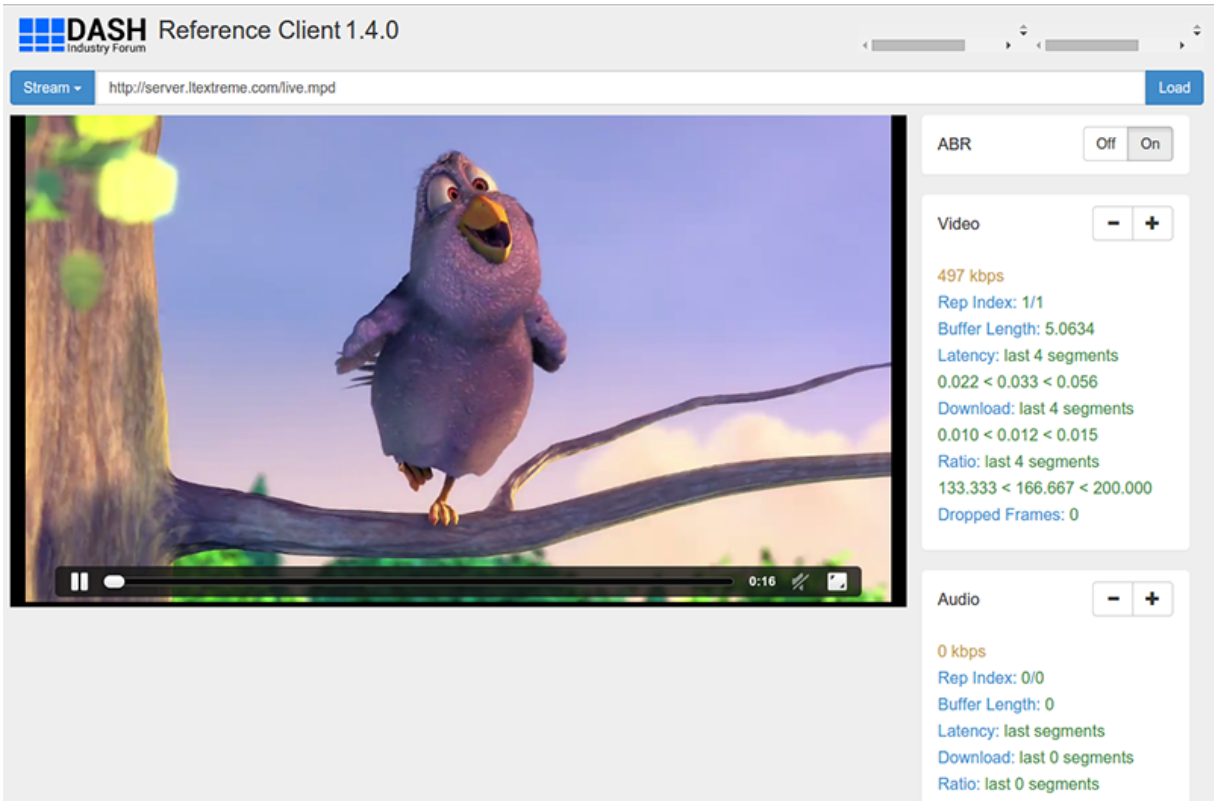

Figura 3.1 : Interfaz gráfica de Dash.js

La principal característica es el aporte de un paquete completo listo para utilizar. Dicho paquete nos proporciona una página HTML (Figura 3.1) definida que contiene el reproductor multimedia y el código Javascript que ejecuta DASH. Por lo tanto, para hacer uso de ello en nuestra maqueta basta con proporcionárselo al servidor web Apache del componente Server para que los usuarios accedan a él, se lo descarguen y así poder disfrutar del streaming de vídeo. Esta página web es bastante sencilla para el usuario dado que únicamente se necesita proporcionar la URL del fichero MPD para un determinado streaming. Esto se puede hacer tanto introduciendo la dirección manualmente o buscando la emisión deseada dentro de la lista precargada, donde también incluiremos como opción el MPD de nuestros experimentos Live.

3.6 VLC Plugin

El reproductor multimedia VLC [16] tiene un nuevo plugin que permite utilizar el programa como un cliente DASH. Simplemente configuramos parámetros como el tamaño deseado de la imagen y posteriormente indicamos la URL del MPD del streaming en cuestión.

4. MAQUETA LIVE STREAMING

4.1 Arquitectura

La arquitectura definida por el 3GPP en el capítulo 2, se ha simplificado enormente comparado con un escenario real, que sería el mostrado en la figura 4.1a donde encontramos los principales componentes que forman la red LTE en coordinación con una red CDN (Content Delivery Network) encargada de distribuir el contenido multimedia generado en directo. Así pues, se utilizará un escenario reducido haciendo uso de un total de cuatro máquinas virtuales más el host como se puede ver en la figura 4.1b.

La primera máquina se corresponde con un servidor de contenidos nombrado Server, cuya función es la de generar y preparar el contenido multimedia del streaming, troceándolo en segmentos, para posteriormente ser enviados a la red. El siguiente elemento que encontramos se conoce como Shaper, componente encargado de simular el comportamiento de la red LTE, pudiendo modificar las características de los enlaces a los clientes asemejándose a los resultados obtenidos con un enlace radio real. Por último, se encuentran conectadas al Shaper dos máquinas virtuales llamadas Client 1 y Client 2, jugando el papel de los usuarios que accederán al servicio de streaming y lo visualizarán haciendo uso de sus smartphones. Debido a las limitaciones de la virtualización en las máquinas virtuales de cada cliente no hay interfaz gráfica, así pues, gracias al host podremos acceder a cada una y observar el streaming de vídeo recibido por el dispositivo.

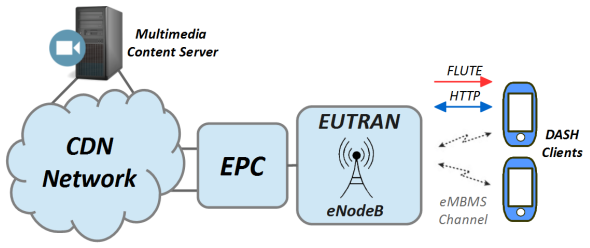

Figura 4.1a : Escenario real

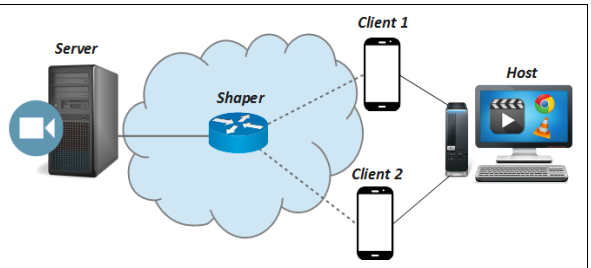

Figura 4.1b : Maqueta Live Streaming

En la siguiente figura 4.2 se muestra el esquema resultante proporcionado por la herramienta VNX tras el despliegue del escenario virtualizado. De acuerdo a lo que se ha explicado sobre el funcionamiento de VNX en el capítulo 3.1, podemos observar que cada uno de los componentes que forman parte de la maqueta son contenedores ligeros LXC.

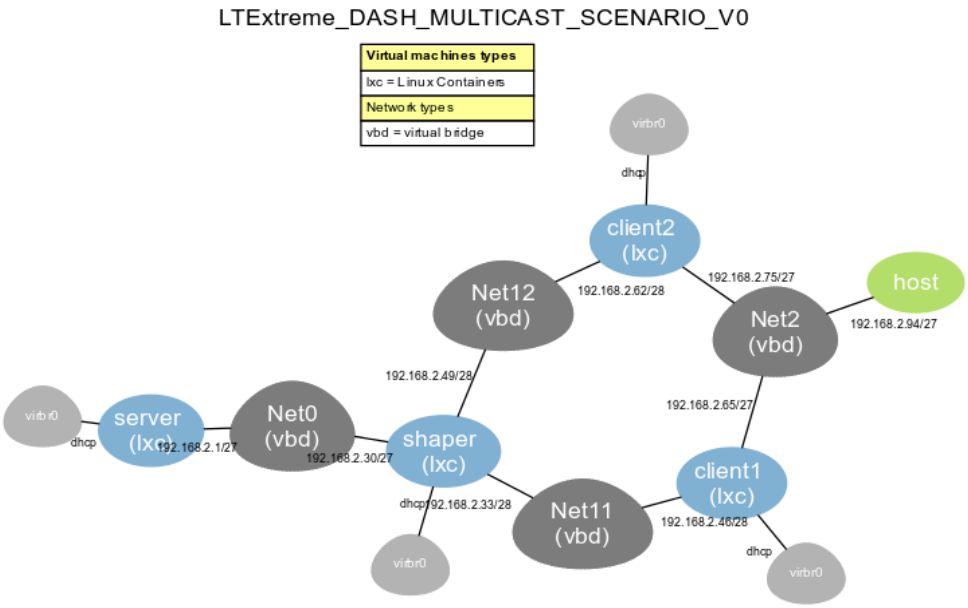

Figura 4.2 : Esquema de la red virtualizada

Esta figura generada por VNX es de gran utilidad ya que describe muy bien la composición de la arquitectura construida con los valores de las interfaces de red para cada máquina virtual.

4.1.1 Addressing

Inicialmente la arquitectura de red de la maqueta LTE constaba únicamente de un cliente. Por este motivo dentro del rango de prefijos (Figura 4.3) simplemente se contemplaban las siguientes tres subredes privadas:

- ***Net 0:*** *192.168.2.0 / 27*
- ***Net 1:*** *192.168.2.32 / 27*
- ***Net 2:*** *192.168.2.64 / 27*

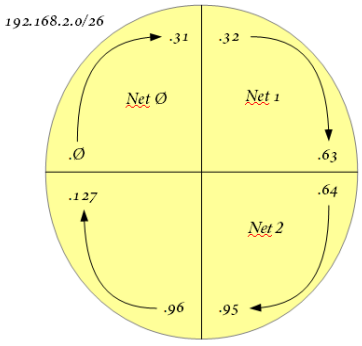

Figura 4.3 : Addressing con un cliente

No obstante, tal como se puede apreciar en la figura 4.1, donde se muestra la topología completa que se ha empleado para el proyecto de investigación, se han utilizado un total de dos clientes para poder estudiar la transmisión multicast que será explicada en detalle en posteriores capítulos. Una posible solución para introducir al segundo cliente en el escenario podría haber sido asignarle una interfaz dentro del rango de direcciones de Net 1 de tal modo que ambos clientes compartiesen la misma red. Sin

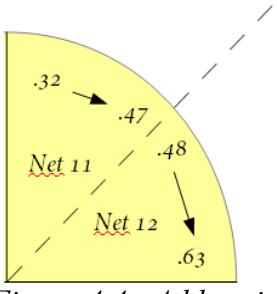

Figura 4.4 : Addressing con dos clientes

embargo esta implementación no es nada deseable dado que en realidad los usuarios no accederán al Core de la red a través del mismo enlance. Cada smartphone se conectará a la red a través de enlaces radio con características diferentes según su posición con respecto al eNodeB. Además, tal como se detallará, dado el funcionamiento del Shaper necesitamos individualizar las conexiones de los clientes a este. Por lo tanto, se decidió dividir [17] la subred Net1 en dos subredes Net11 y Net12 mediante las cuales accederán al núcleo de la red los clientes 1 y 2 respectivamente. En la figura 4.4 se muestra el resultado final de la representación de prefijos para dicha división, de modo que las subredes definitivas operativas en nuestro escenario son las siguientes:

- ***Net 0:*** *192.168.2.0 / 27*
- ***Net 11:*** *192.168.2.32 / 28*
- ***Net 12:*** *192.168.2.48 / 28*
- ***Net 2:*** *192.168.2.64 / 27*

4.1.2 Componentes

Server

En uno de los extremos de la comunicación tenemos el Server, el cual se va a encargar de distribuir los contenidos a los clientes. Este servidor multimedia es el elemento estrella dentro del servicio de vídeo que deseamos testear en nuestro escenario. Los mecanismos que se van seguir en su interior son clave tanto para el envío de datos como para la posterior visualización del contenido multimedia en los dispositivos de los usuarios. Con respecto al transporte de los segmentos generados se va a hacer uso de una implementación del protocolo FLUTE llamada Project MAD Flute, y que nos va a permitir enviar tráfico multicast al núcleo de la red para que lo reciban aquellos usuarios interesados. Por otro lado, el servidor tiene instalado un servidor web Apache cuya función será la de ofrecer una página web con un reproductor en el cual se encuentra el cliente que implementa el protocolo DASH.

Shaper

El segundo elemento que se nos presenta en el escenario es el llamado "Shaper". Dicho componente podría llegar a tener una enorme complejidad ya que su función es la de simular el comportamiento de la red. Idealmente en esta virtualización habría que imitar el funcionamiento de la arquitectura Evolved Packet System (EPS) de la red LTE. Esto significaría tener en cuenta por un lado la comunicación entre la red de acceso Evolved UTRAN (E-UTRAN) con el núcleo de la red conocido como Evolved Packet Core (EPC); mientras que por otro lado los efectos asociados al enlace radio utilizado en la conexión del smartphone con el eNodeB de la red de acceso.

Sin embargo, la simulación de todos estos factores no es el objeto de estudio de este trabajo, así pues se ha optado por simplificar al máximo el funcionamiento de este componente que representa a toda la red. De este modo, el Shaper simplemente va a tener la capacidad de asociar a sus respectivas interfaces diferentes características que puedan limitar las comunicaciones como pueden ser el bandwidth, el retardo, el jitter o la probabilidad de perder paquetes. Esta funcionalidad se incorpora a la máquina a través de una serie de scripts basados en el uso de una disciplina Class-based queuing (CBQ) que clasifica el tráfico entrante, y la herramienta Linux Traffic Control (TC) junto con la herramienta Network Emulation (NetEm) para controlar el ancho de banda, retardo y pérdidas. En total hay dos scripts principales, "shaperf" y "shaperc", donde ambos tienen la misma función de procesar el tráfico egress, pero uno con parámetros que pueden variar con el paso del tiempo mientras que el otro toma unos parámetros iniciales que no van a cambiar salvo que cortemos dicho control. Esto se consigue mediante el uso de un simple fichero de texto en el que se indica las características de la red.

Clients

El componente Client se puede considerar como el terminal LTE y está compuesto por un cliente DASH, un cliente FLUTE y una caché HTTP intermedia. Para el desarrollo de las pruebas el terminal se ha dividido en dos partes: una máquina virtual que ejecuta el cliente FLUTE y una caché HTTP, y por otro lado un cliente DASH funcionando fuera del entorno virtualizado que solicita el contenido de vídeo debido a limitaciones gráficas de la virtualización ligera.

Host

El último elemento que conforma toda nuestra maqueta es el propio host con OS Linux sobre el que estamos ejecutando VNX para virtualizar nuestra maqueta al completo. En un caso real únicamente existiría el smartphone, de modo que incluiría todas las funcionalidades del lado del cliente como son la comunicación con el server y la visualización del vídeo. Sin embargo, en la maqueta el Client es un contenedor virtual que carece de GUI, así pues, nuestro Host accederá a la web hospedada por el Server para obtener el cliente javascript que implementa y coordinándose con la chaché Squid podremos reproducir y visualizar el vídeo en nuestro PC. Para ello simplemente nos apollaremos en la capacidad de Google Chrome para el acceso web a través del proxy que le indiquemos, en este caso el Client1 o Client2. En resumen, la función del host equivaldría a la segunda mitad que conformaría el smartphone de uno de los dos clientes.

4.1.3 Multicast

Hay una funcionalidad crucial que debe soportar nuestra red dada las características del servicio que pretendemos ofertar, y esta es la transmisión de datos en multicast. Para poder

ofrecer soporte a este tipo de envío se ha decido utilizar el protocolo IGMP [18], el cual nos permite poder gestionar los estados de las asociaciones dinámicas que existan entre un router, que se encontraría en el Shaper, y los clientes que formen parte del grupo de multidifusión. No obstante, con respecto al router que hemos mencionado existen dos variantes de IGMP que cabe la posibilidad aplicar: Snooping y Proxing.

Figura 4.5 : IGMP Snooping

Para poder resolver esta cuestión debemos pensar por un momento en el número de potenciales clientes para nuestro servicio de vídeo. Estamos hablando de miles de usuarios, por lo que habrá una enorme cantidad de conexiones con el Shaper. Así pues, como podemos observar en la Figura 4.5, si optásemos por utilizar una configuración IGMP Snooping inundaríamos una red plagada de usuarios con peticiones IGMP Request ya que los clientes se comunicarían directamente con el servidor dejando al router del Shaper completamente al margen. Por este motivo esta solución no sería correcta, ya que entre el servidor y cada usuario habría una conversación y dado el enorme número de clientes no sería nada eficiente. En cambio, si nos fijamos en el funcionamiento del IGMP Proxy en la Figura 4.6, dotamos de inteligencia a nuestro nodo de modo que actúe de intermediario entre el Server y los clientes. Con este mecanismo mejora sumamente la eficiencia ya que entre el servidor y el router existirá un único flujo IGMP, y entre el Shaper y los clientes flujos con sólo aquellos que se asocien al grupo multicast. Recordemos que este modelo es utilizado en otros escenarios muy comunes, como puede ser en Movistar Fusion TV para la difusión de televisión a los hogares de los usuarios.

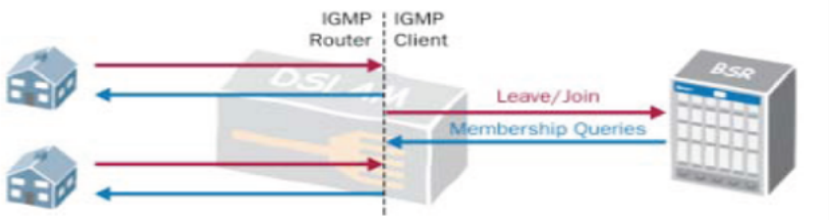

Figura 4.6 : IGMP Proxy

Por lo tanto, de acuerdo con lo que hemos explicado se ha optado por introducir la configuración IGMP Proxy en nuestro nodo. Dicha implementación se ha llevado a cabo mediante la combinación de un pequeño programa llamado *smcroute* y otro de mayor envergadura llamado *mcproxy* [19]. Empezaremos centrándonos en el segundo ya que el primero se utiliza en los clientes. La puesta en marcha de mcproxy previamente requiere que nos cercioremos sobre la habilitación para manejo de tráfico IGMPv2 en todas las máquinas del escenario. Posteriormente el proceso se divide en tres simples pasos:

1. Modificación del fichero de configuración en el cual se indica al nodo a través de qué interfaz de red le llega el flujo de datos para un determinado grupo multicast y por otro lado, las interfaces de red por las que escuchar IGMP Queries. Este fichero se puede encontrar al completo en el Anexo B.

2. Activación del programa para detectar las tablas de forwarding que contenga.

3. Activación del programa pasándole como parámetro nuestro fichero de configuración. Tras la ejecución del comando se mostrará por el terminal en tiempo real el estado sobre la información (Figura 4.7) que contiene el nodo con respecto a a los grupos multicast gestionados.

En la siguiente figura se muestran las trazas generadas por mcproxy, en las cuales se puede apreciar el estado para el grupo multicast 225.1.2.3 recibiendo un flujo entrante procedente del servidor de contenidos y reenviando dicho flujo por interfaz de red eth2, ya que únicamente se encuentra asociado al grupo multicast el Client 1.

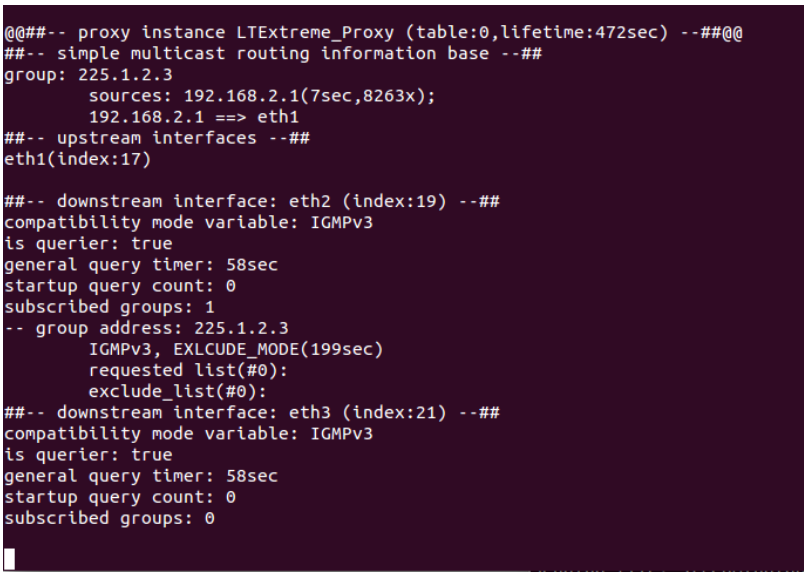

Figura 4.7: Trazas del programa mcproxy

Una vez establecido nuestro puente intermedio en la red para reenvíar el tráfico multicast nos vamos a centrar en detallar el granito de arena que pone cada uno de los extremos, tanto el Server como el Client. En primer lugar debemos tener en cuenta que las direcciones IP Multicast son de clase D, es decir, están comprendidas entre 224.0.0.0 y 239.255.255.2555. No obstante, atendiendo a la documentación CISCO [20] donde se muestra los rangos de direcciones establecidos por IANA con sus respectivos propósitos, se ha optado por utilizar como Multicast Group IP la dirección 225.1.2.3 dado que se encuentra en un rango liberado. Así pues, en el Server ha de añadirse una nueva ruta 224.0.0.0/4 en su tabla de

forwarding puesto que antes no era capaz de manejar este tráfico, y de esta forma ya estará capacitado para enviar correctamente los segmentos del streaming de vídeo a la dirección IP 225.1.2.3, de la cual el Proxy IGMP se encargará de publicar a los clientes.

En segundo lugar, observando el esquema de la arquitectura de red en la figura 4.1 encontramos un total de dos clientes. Tal como se ha mencionado, inicialmente en nuestro escenario existía un único usuario pero se decidió añadir un segundo para asemejar la situación más a la realidad al poder ofrecer el servicio de streaming con envío multicast. Con el objetivo de dar soporte a esta funcionalidad, en los dispositivos de los usuarios se va a hacer uso del pequeño programa llamado *smcroute* el cual nos permite realizar queries de tipo IGMP Join o IGMP Leave. Este programa se fundamenta en la ejecución en paralelo de un demonio encargado de la comunicación IGMP.

Smcroute se trata del último engranaje que va a conseguir una perfecta transmisión multicast entre el servidor y los clientes del servicio de vídeo. El mecanismo a seguir en cada cliente es muy sencillo y basta con utilizar un simple comando para realizar una petición IGMP Join al unirnos al streaming y una petición IGMP Leave cuando deseemos abandonar la sesión.

4.2 Implementación de FLUTE

4.2.1 Implementación

Ahora que ya conocemos los mecanismos que utiliza el protocolo FLUTE podemos abordar en detalle cómo se ha implementado en nuestro proyecto de investigación. Dicha implementación se ha llevado a cabo con el objetivo que se propone el servicio de streaming de vídeo que pretendemos ofrecer. Dada la naturaleza de FLUTE para el envío de datos multicast, se trata un solución idónea para realizar la distribución de los segmentos, en los que dividimos nuestro vídeo, a un numeroso grupo de usuarios en la red.

De esta forma podremos iniciar una sesión de FLUTE y el posterior envío de objetos descritos en las instancias FDT asociadas a la sesión con el orden que se indica en la imagen. Además, este tráfico multicast no inundará la red ya que el flujo procedente del servidor se detendrá en el proxy IGMP del Shaper y únicamente será reenviado por aquellas interfaces de red por las que existan clientes unidos al grupo de difusión. Por otro lado, gracias a que la conexión utilizada por FLUTE es unidireccional, concretamente del servidor a los usuarios del servicio, los clientes podrán unirse o abandonar la sesión FLUTE dinámicamente. En otras palabras, un cliente no recibirá los datos enviados por el servidor hasta que se una al grupo multicast, y posteriormente dejará de recibirlos una vez abandone el grupo y a su vez se desconecte de la sesión FLUTE.

4.2.1.1 Flujos conceptuales

Sin embargo, para conseguir la implementación completa de FLUTE y construir con éxito nuestro servicio de streaming no se ha requerido sólo de la aplicación MAD-FLUTE y sus librerías correspondientes, sino también de la coordinación de una serie de scripts instalados tanto en el Server como en los Clients. En las siguientes figuras se muestran mediante unos flujos conceptuales el funcionamiento de cada script escrito en Shell.

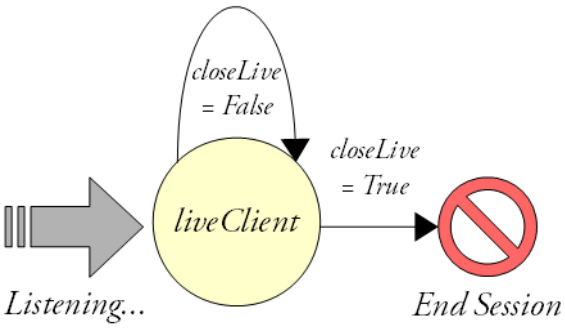

Figura 4.8: Esquema liveClient

En el caso del cliente se necesita un único script llamado "liveClient" que va a pasar por un total de tres estados (Figura 4.8). En primer lugar se lleva a cabo una asociación con el grupo multicast de la sesión realizando una petición IGMP Join al nodo del Shaper. El segundo estado será la recepción de la sesión FLUTE, y en tercer y último lugar, el abandono de la sesión FLUTE y del grupo multicast. Este abandono se realizará cuando el cliente reciba el fichero "closeLive".

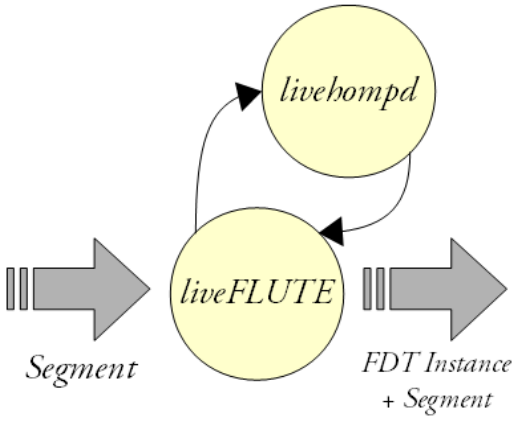

Figura 4.9: Esquema liveFLUTE + livehompd

Por otro lado tenemos el esquema correspondiente al Server. En esta ocasión el funcionamiento es más complejo dado que necesitamos de la coordinación de dos scripts para proporcionar el servicio. Como se puede observar en la figura 4.9 encontramos el llamado "livehompd" y "liveFLUTE". Podemos definir a grandes rasgos al script liveFLUTE como el "main thread" ejecutado en el servidor. Su función va a ser, tomando como parámetro de

entrada un directorio donde se disponen de los segmentos de vídeo, iniciar una sesión FLUTE y enviar todos estos ficheros hasta que no queden más. No obstante hay que matizar una serie de características especiales que hemos introducido con el fin de asemejar la situación a un escenario real.

1. La primera característica es que se trata de una transmisión de vídeo en directo, por tanto no vamos a disponer de todos los segmentos desde el principio. Así pues aquí interviene el script livehompd, cuya función es la de crear un FDT actualizado para los nuevos segmentos generados en vivo y devolvérselo de nuevo a liveFLUTE para que envíe la nueva instancia FDT por la sesión de FLUTE a los clientes.

2. Otra estrechamente relacionada con la anterior es debida a la división temporal que se ha realizado en los segmentos. Puesto que todos duran dos segundos, para imitar a un caso real tenemos que pensar en que la cámara grabará durante dos segundos, codificará y generará el segmento de vídeo listo para ser enviado. Así pues, nosotros vamos a introducir una espera de dos segundos, acto seguido generamos el FDT actualizado con este segmento y posteriormente enviamos por la sesión FLUTE la nueva instancia y el nuevo segmento.

3. Estos nuevos segmentos de vídeo recién generados van a ser replicados en otro directorio tras la espera de dos segundos. Los motivos de está solución se explicarán con detalle en el capítulo 5.

4. Por último, con el fin de automatizar lo más posible la transmisión, se ha establecido que en el servidor si no se detectan nuevos segmentos generados significa que el streaming ha concluido. Por lo tanto, la medida que se ha tomado es que tras esta detección se lleva a cabo una espera de 5 segundos y se envía a los clientes el fichero "closeLive" que les informa del fin del streaming.

4.2.1.2 Problemas asociados y resultado final

Sin embargo estamos hablando de un protocolo que apenas ha evolucionado en los últimos 10 años. A su vez, el desarrollo de la herramienta MAD-FLUTE que hemos utilizado para implementar el protocolo se encuentra dentro del paquete MAD-FCL, del cual se lanzó su última release en Marzo de 2007, por lo que también ha dejado de ser mantenido y revisado en los últimos años. Este hecho significa que hay una alta probabilidad de enfrentarse a obstáculos, y tras haber exprimido al máximo la herramienta podemos decir que es cierto. A continuación, detallamos los problemas a los que nos hemos enfrentado y las correspondientes soluciones que hemos decidido optar con el objetivo de dar el servicio de streaming en directo en multicast.

Para poner en situación supongamos que el objetivo es emitir un partido de fútbol en directo, así pues la idea es desde el servidor iniciar una sesión FLUTE para todo el streaming a la que los usuarios se van a unir y abandonar dinámicamente. Dado que se trata de una transmisión de segmentos "al vuelo" tendremos que realizar un envío de instancias FDT con los segmentos nuevos, de modo que la base de datos FDT del cliente se vaya actualizando durante el transcurso de la transmisión. Concretamente prepararemos y enviaremos una nueva instancia FDT por cada segmento nuevo generado tal como se muestra en el esquema de la figura 4.10.

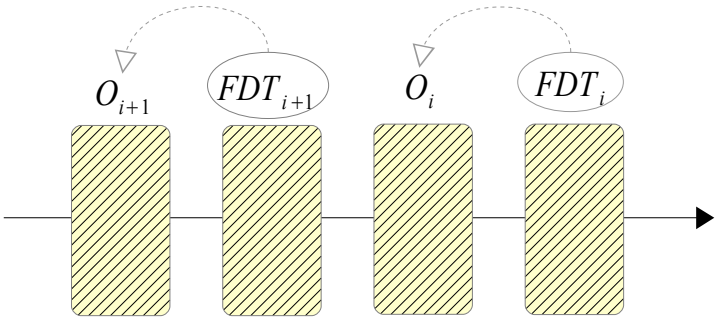

Figura 4.10: Orden de transmisión en FLUTE

Esta funcionalidad conlleva tener en cuenta una serie de obligaciones tal como se marcan en el estándar de FLUTE. Entre ellas destacamos en este caso que cada segmento nuevo debe tener asociado un TOI único, puesto que si enviamos el segmento con un TOI=2 y el cliente está esperando un TOI > 2, lo descartará al recibirlo. Otra norma de FLUTE de enorme importancia es, tal como hemos mencionado en capítulos anteriores, cada instancia FDT nueva debe llevar un ID superior al anterior para poder actualizar la base de datos FDT del cliente. Es decir, observando la figura, si la última instancia tenía un ID=2 para el TOI=1, la nueva que enviamos tendrá que tener un ID=3 para el TOI=2. Si no fuese así, el cliente detectaría que es la instancia no es nueva y la tiraría, y consecuentemente descartaría después el segmento con TOI=2.

Relativa a esta última sentencia se nos presenta el mayor problema de la implementación del protocolo FLUTE en el proyecto. La herramienta MAD-FLUTE envía siempre las instancias con una ID=0, por lo que la base de datos FDT en la aplicación del cliente descarta todas las instancias FDT tras la primera recibida. Además, tal como se indica en el estándar de FLUTE, el FDT Instance ID lo lleva el paquete en la cabecera EXT_FDT y dicho valor no se debe reutilizar para instancias que no han expirado. Eso conlleva a un error o el directo abandono de la sesión". El problema de la aplicación es que no tiene la capacidad de acceder a las cabeceras, y por tanto no se puede modificar el valor de la cabecera EXT_FDT. Por lo tanto, a partir de este punto muerto si nos ceñimos a las posibilidades que nos aporta FLUTE según la RFC, se nos abren varios caminos para encontrar una solución.

En la referencia anterior hablan de instancias que no han expirado, así pues, una posible opción es reutilizar siempre el FDT Instance ID=0 consiguiendo que la última

instancia que recibió el cliente caduque antes recibir una nueva. Esta medida supone llevar un control exhaustivo del tiempo de expiración de las instancias FDT, así como de la necesaria sincronización del reloj de todos los componentes de la red. Con respecto a la sincronización no hay problema alguno dado que la virtualización proporcionada por VNX conlleva que todos los contenedores compartan el kernel del host, y por tanto, compartan el mismo reloj. En segundo lugar, para los tiempos de expiración de la instancias generó muchos problemas debido al modo en que se había implementado FLUTE en el paquete MAD-FCL. Tras una gran inversión de horas investigando se detectó en el código C++ de la aplicación que los desarrolladores suponían lo siguiente:

```
01469   while(get_session_state(receiver->s_id) == SActive) {
01470
01471     time(&systime);
01472     curr_time = systime + 2208988800U;
01473
01474     /* Get initial fdt */
01475     if(receiver->fdt == NULL) {
```

Figura 4.11: Primer fragmento del código receiver.c

Asumen que el reloj de la sesión FLUTE es, "*curr_time = systime + 2208988800" (Figura 4.11).* Esto directamente significa que la instancia no va expirar en años, por que no sería posible reutilizar lD=0 en la base de datos del cliente. Para mitigar este obstáculo, se trampeó sumando a nuestro reloj interno esa enorme cantidad de tiempo en formato NTP para sincronizar ambos al rellenar el valor "expire time" en el fichero XML de la instancia FDT. Sin embargo, se demostró que la herramienta no era capaz de reutilizar FDT Instance IDs. Para cerciorarnos de ello se provocó que expirase la instancia al segundo de enviarse desde del server, de modo que llegase al cliente caducada. El resultado efectivamente fue que el cliente descartaba esa instancia, así que al menos esta característica de FLUTE la cumplía.

```
01515       if(fdt_instance->expires < curr_time) {
01516
01517         if(!receiver->accept_expired_fdt_inst) {
01518
01519           if(receiver->verbosity == 4) {
01520             printf("Expired FDT Instance received, discarding\n");
01521             fflush(stdout);
01522           }
01523
01524           free(buf);
01525           FreeFDT(fdt_instance);
01526           continue;
01527         }
01528         else {
01529           if(receiver->verbosity == 4) {
01530             printf("Expired FDT Instance received, using it anyway\n");
01531             fflush(stdout);
01532           }
01533         }
01534       }
```

Figura 4.12: Segundo fragmento del código receiver.c

El problema es que tal como está desarrollada la aplicación, si intentamos realizar la comprobación anterior parece ser que en la base de datos FDT del cliente la instancias no caducan, por lo que no es posible reciclar el ID de la instancia para futuras. Si observamos en

la figura 4.12 se hace la comprobación de la que estamos hablando, sin embargo, inexplicablemente la ejecución del programa no es capaz de entrar en el if principal al no mostrar ninguna de las dos posibles trazas y debería ser capaz ya que nos hemos cerciorado de que el valor utilizado puede caducar.

Por consiguiente se plantearon otras posibles soluciones, de las que ninguna llegó a materializarse excepto una, la cual era la más rudimentaria pero conseguía cumplir el objetivo que nos proponíamos de envíar segmentos "al vuelo". A grandes rasgos se puede decir que la solución consistía en crear y cerrar una nueva sesión FLUTE para el envío de cada segmento. Para ello se tuvo que realizar una modificación en la coordinación de los scripts del server y cliente que explicamos en el capítulo 4.2.1.1. El punto clave se encuentra en el mecanismo seguido por el cliente, escuchando en bucle nuevas sesiones FLUTE hasta recibir el fichero llamado "closeLive". Esto supone que debe haber una sincronización perfecta entre ambos componentes, para que le de tiempo al cliente a cerrar sesión, abrir una nueva y recibir la instancia FDT y el objeto. En la figura 4.13 se muestra que esto es posible gracias a que el tiempo de recepción son aproximadamente los dos segundos de grabación del segmento más el retardo de propagación en la red, aunque dicho retardo es nulo dadas las características de nuestra red virtualizada.

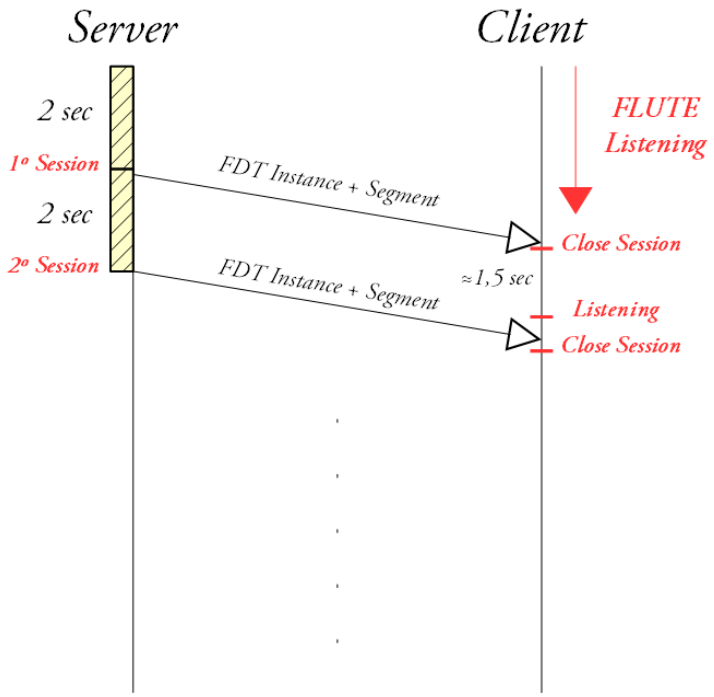

Figura 4.13: Sincronización sesiones FLUTE

Tras numerosas pruebas se estimó que el tiempo medio requerido por la herramienta MAD-FLUTE en recibir la instancia FDT+segmento, cerrar la sesión y abrir una nueva es de aproximadamente unos 1,5 segundos. Por lo tanto, gracias al margen de dos segundos de grabación con el que partimos tenemos garantizada la recepción en el Client. No obstante, como hemos dicho, esta medida era el último recurso que se planteó puesto que nuestro

principal interés era simplemente establecer una única sesión para todo el streaming, aunque nos permite solventar el problema de las instancias FDT ya que de esta forma no puede haber conflictos.

Por último, se debe mencionar que ante tantos obstáculos impuestos por esta aplicación, se intentó instalar en la maqueta una nueva herramienta de implementación de FLUTE desarrollada en Java llamda jFlute [21]. A diferencia de la que hemos utilizado, esta nueva permite modificar el código y hacerlo compatible con cualquier dispositivo al estar escrito en Java. Sin embargo, no se pudo completar la instalación de la aplicación debido a que utiliza una librería que fue escrita en C y compilada en Windows. Se hizo una busqueda intensiva de esa librería por Internet pero está obsoleta, por lo que no fue posible hacer uso de ella en Linux, tal como indican sus desarrolladores en la página web.

4.2.2 Pruebas FEC

4.2.2.1 Motivación

El servicio de streaming de vídeo que deseamos ofrecer se apoya en la tecnología móvil LTE. Por tanto, la transmisión de los datos a los usuarios finales se va a realizar a través de canales radio, lo que supone el uso de enlaces con un cierto grado de limitaciones dependiendo de la posición del usuario con respecto del eNodeB y del número de otros usuarios con acceso al mismo. En otras palabras, dependiendo de la calidad que dispongamos en el enlace radio podremos tener mayor o menor ancho de banda, retardo de propagación o un determinado porcentaje de pérdidas de los paquetes transmitidos. Estas variables pueden provocar que la reproducción se pare en múltiples ocasiones debido a las pérdidas, así como también provocar una calidad del vídeo mala debido a la reducción del ancho de banda.

Con respecto a la calidad del vídeo que nuestros clientes van a poder disfrutar, es el principal objetivo del protocolo DASH, el cual explicaremos posteriormente. Sin embargo, antes de desembocar en DASH, en la transmisión FLUTE podemos intentar mitigar lo máximo posible el efecto negativo que supone las pérdidas de paquetes. Esto se puede materializar gracias a la codificación de los datos que enviamos desde el servidor mediante códigos FEC como puede ser el Reed-Solomon. Dado que no es objeto de estudio de este trabajo, no se entra en detalle a explicar el funcionamiento de estos mecanismos ya que además nos extenderíamos enormemente. No obstante, la herramienta MAD-FLUTE utilizada aporta la capacidad de introducir de forma sencilla una cierta redundancia en los paquetes que enviamos desde el servidor. Esta información extra que añaden los códigos FEC a cada bloque de datos, permite que en caso de que el cliente lo reciba con errores, sea entonces capaz detectarlo y corregirlo. Esto mejoraría enormemente el rendimiento de la reproducción del vídeo al evitar que el usuario solicite retransmisiones unicast por HTTP al servidor debido a que un segmento se ha perdido o corrompido, provocando que la visualización pueda llegar a interrumpirse.

4.2.2.2 Análisis

De acuerdo a lo detallado en el capítulo introductorio de los componentes de la maqueta, recordemos que en el Shaper tenemos la capacidad de modificar las características del tráfico egress en las interfaces de red con destino los clientes. Concretamente para asemejarnos a una situación real, idealmente se haría uso del script "shaperf". Es el idóneo puesto que permite modificar dinámicamente con el transcurso del tiempo las características del enlace que lo conectan con el usuario, ya que en un escenario real el cliente puede cambiar de posición con respecto a la antena. Sin embargo, para nuestro objetivo esta modalidad no nos interesa porque la meta es analizar las ventajas que nos aportan los códigos FEC sin introducir una excesiva redundancia.

Por este motivo se decide utilizar el script "shaperc", que establece valores fijos tanto para el ancho de banda disponible, el retardo introducido por la red y el porcentaje de pérdidas sufrido por los paquetes. En especial, es esta última variable con la que jugaremos en las pruebas realizadas, con la finalidad de detectar cuál es el "code rate" necesario para conseguir un porcentaje aceptable de segmentos perdidos en la transmisión. El llamado code rate se trata de una relación que expresa el porcentaje de información útil contenida en cada bloque de datos enviado. Para nuestro caso de uso, se sabe que la herramienta toma por defecto un valor para el "Maximum-Source-Block-Length" de 64. Este parámetro representa el número máximo de símbolos fuente que puede contener un bloque fuente, es decir, la información útil. Partiendo de esta información se obtiene el parámetro "Max-Number-of-Encoding-Symbols". Este es el número de símbolos codificados a enviar en un bloque y cuyo valor depende también también del porcentaje de redundancia que añadamos. De esta forma podemos definir el code rate utilizado por la herramienta mediante la relación mostrada en la figura 4.14.

$$X' = \frac{X}{100} + 1 \qquad TOTAL = 64 * X'$$

Figura 4.14: Fórmula cálculo de redundancia

En la imagen se muestra la operación que nos proporciona el valor del número total de símbolos codificados a partir del número de símbolos fuente en un bloque transmitido. El valor que indicamos con la incógnita X es introducido por comandos de la herramienta MAD-FLUTE al iniciar la sesión desde el servidor. Dicha variable en el contexto de la aplicación es llamada "FEC-Ratio" y su valor es equivalente al porcentaje de redundancia que queremos añadir. Por último cabe destacar que se ha optado por utilizar el código FEC Reed-Solomon (old I-D), que entre los proporcionados por la herramienta es el más robusto y estable de todos.

4.2.2.3 Resultados

El procedimiento seguido en las pruebas ha consistido en introducir un porcentaje de pérdidas fijo en el enlace de red, y para distintos FEC Ratios se ha medido el número total de segmentos perdidos en la transmisión. Puesto que el vídeo utilizado en la maqueta tiene una duración de 10 minutos y se ha divido en trozos de dos segundos, son enviados un total de 300 segmentos en cada prueba. Con el fin de obtener un resultado preciso para cada porcentaje de pérdidas y FEC Ratio, se repite la prueba tres veces y se anota el valor medio. Los resultados obtenidos se recogen en la siguiente figura.

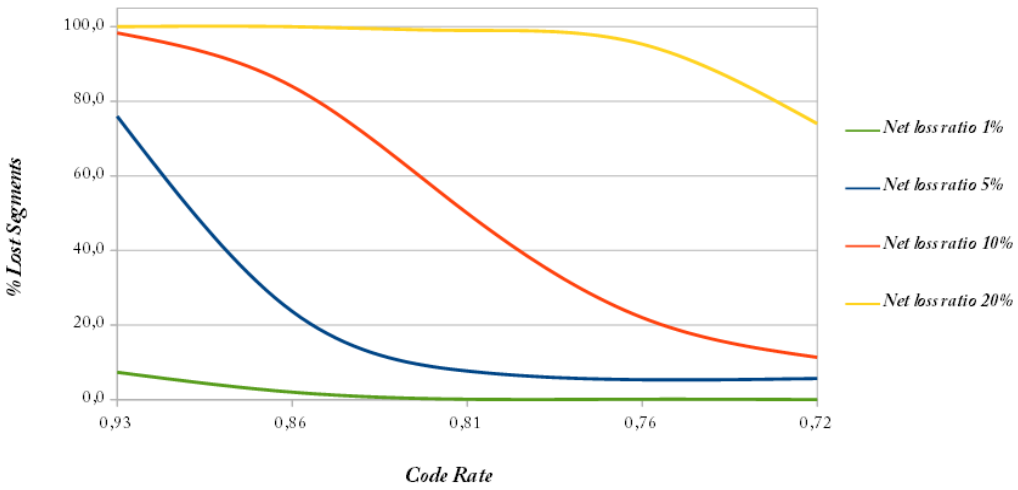

Figura 4.15: Gráfica con resultados de las pruebas con FEC

En la figura 4.15 podemos apreciar que estamos hablando de un código antiguo. Requiere añadir una gran redundancia para poder ofrecer resultados óptimos, siempre y cuando no sobrepasemos el umbral del 15%-20% de pérdidas en la red. Como se puede observar, en las pruebas sólo se ha considerado un code rate entre 1 y 0.7 debido a que valores inferiores son muy poco eficientes. El hecho de introducir información extra nos perjudica enormemente puesto que nos interesan transmisiones de segmentos lo más rápidas posibles, puesto que de este modo estamos reduciendo el caudal efectivo.

Así pues se puede obtener como conclusión que nuestra herramienta MAD-FLUTE cojea bastante de este pie. Intentar implementarla en un caso real en el que tengamos un canal radio de mala calidad supondría tener un porcentaje desmesurado de pérdidas de segmentos en la transmisión. Este gran obstáculo es otra de las grandes motivaciones por las que se hizo una búsqueda de otras aplicaciones que implementasen el protocolo FLUTE. Por ejemplo, si hubiese sido posible la instalación de la herramienta jFlute que ya hemos mencionado, al estar escrita en Java, se podrían haber añadido nuevas librerías que permitirían hacer uso de los

códigos Raptor. De esta forma se hubiesen conseguido resultados mucho mejores en estas pruebas ya que aportan un rendimiento mayor al Reed-Solomon, y así realmente poder hacer frente a las peculiaridades de los canales radio.

4.3 Implementación de DASH

4.3.1 Propiedades elementos XML

Tal como se explicó en el capítulo 2.4, DASH requiere de un fichero MPD en el cual se definen las distintas representaciones del streaming que se está transmitiendo. Este fichero, escrito en XML, es descargado por el cliente al unirse al streaming para saber cómo está distribuido el contenido multimedia y cómo ha de ser accedido. La clasificación de esta información sigue un modelo de datos jerárquico que se podrá reflejar fácilmente mediante el uso de las etiquetas caracterisiticas de XML. A la hora de construir este fichero en cuestión, cada elemento XML representa cada uno de los niveles de la jerarquía que hablamos como puede ser Representation o Adaptation Set. A su vez, estos elementos XML van a tener una serie de propiedades que darán lugar finalmente a la definición completa del streaming. A continuación, para cada elemento presentaremos en detalle las propiedades más importantes ya que serán de enorme importancia en la realización de los experimentos en directo.

4.3.1.1 MPD

El primer elemento que podemos indentificar en el documento es el MPD. Se puede considerar como la raíz de la Media Presentation Description.

- **@type :** Propiedad de crucial importancia. Con el valor "static" indica que todos los segmentos están disponibles en el servidor y pueden ser solicitados. Con el valor "dynamic" establecemos un streaming en directo, por lo que nuevos segmentos del vídeo irán apareciendo dinámicamente a lo largo del tiempo.

- **@availabilityStartTime :** Fecha a partir de la cual los segmentos empiezan a estar disponibles. Inicia el timeline de disponibilidad de segmentos para el tipo "dynamic". En ese caso, si suponemos que la duración de los segmentos es de dos segundos, durante ese intervalo de tiempo estará disponible y al finalizarlo, estará disponible el siguiente segmento durante un mismo intervalo. En la siguiente figura 4.16 se muestra un ejemplo de la disponibilidad de segmentos en directo en el trancurso del tiempo.

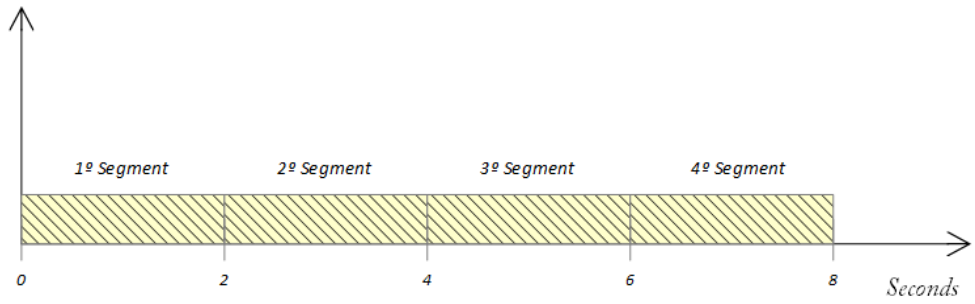

Figura 4.16: Disponibilidad de segmentos en el tiempo

Supongamos que el valor de @availabilityStartTime es el instante cero. Puesto que la duración de los segmentos es de dos segundos, entre los instantes cero y dos el primer segmento estará disponible y DASH intentará solicitarlo para reproducirlo. A partir del instante dos el primer segmento deja de ser accesible y en esta ocasión hasta el instante 4 estará disponible el segundo segmento del vídeo. Este proceso se seguirá así sucesivamente hasta el final del vídeo.

- **@publishTime :** Propiedad de gran importancia cuando el tipo es "dynamic". Su valor es la fecha en la que se generó el último fichero MPD en el servidor de contenidos. Por tanto, supongamos que si el cliente descargó el primer MPD con un @publishTime igual a las 09:00 am, y posteriormente descarga uno en el que el valor de @publishTime ha cambiado a las 09:05 am, esto indica que el segundo MPD descargado es una actualización de la descripción del streaming que está reproduciendo.

- **@mediaPresentationDuration :** Representa la duración total de la presentación multimedia. Así pues, la utilización o no de esta propiedad dependerá de si conocemos cuanto dura el vídeo que estamos emitiendo. Como veremos en los experimentos, para una transimisión en directo este valor es dificil de estimar con exactitud.

- **@minBufferTime :** Especifica la cantidad mínima que debe almacenar el reproductor del vídeo antes de iniciar la visualización.

- **@minUpdatePeriod :** Periodo de tiempo tras el cual el cliente solicita de nuevo al servidor el fichero MPD del streaming. La finalidad de esta petición es detectar si se han producidos cambios en la descripción del contenido, como por ejemplo una futura introducción de un anuncio en medio de la reproducción.

4.3.1.2 Period

Una presentación multimedia se divide en el tiempo en uno o varios Periods.

- **@start :** Propiedad que especifica en el tiempo el valor del PeriodStart. Este valor actúa como el punto de partida para conocer el instante de inical de cada segmento así como el instante de acceso en el timeline de la presentación multimedia.

- **@duration :** Representa la duración total del periodo. Además sirve para calcular el valor PeriodStart del siguiente Period.

Por lo tanto, a la hora de concatenar una serie de periodos el valor del PeriodStart del siguiente periodo vendrá dado por la suma de @start y @duration del anterior periodo. Otra consideración es que si el Period se trata del último de la reproducción y carece del atributo @duration, tomará como instante final del Period el valor del atributo @mediaPresentationDuration especificado en el MPD.

4.3.1.3 Adaptation Set

Un Period está compuesto por uno o varios Adaptation Set. De acuerdo a lo explicado en el capítulo 2.4 recordemos que este elemento consiste en en un grupo de múltiples representaciones posibles de vídeo o audio por ejemplo. Para nuestros experimentos veremos que únicamente utilizamos un Adaptation Set en cada Period ya que sólo transmitimos una única señal de vídeo.

- **@contentType :** Especifica el tipo de contenido multimedia que representa el Adaptation Set en cuestión. El uso de este valor es totalmente recomendable ya que se le facilita al reproductor multimedia el tipo de contenido al que está accediendo.

4.3.1.4 Representation

Agrupados en Adaptation Sets, las Representations son las posibles opciones codificadas para cada componente del contenido multimedia a lo largo de un Period. Por lo tanto, una Representation seleccionada comienza en el PeriodStart y continúa hasta el final de dicho Period.

- **@ id :** Propiedad que permite identificar a una Representation dentro de un Period. Este identificador debe ser único salvo que existan otras Representations dentro de un mismo Period con la misma funcionalidad.

- **@ bandwidth :** Si el ancho de banda que se tiene disponible durante el streaming tiene este valor, DASH tras una serie de cálculos, asignará esta Representation como la adecuada a las capacidades del cliente.

4.3.1.5 Segment

La definición de los segmentos que componen cada Representation se puede realizar de distintas formas. Una muy simple es utilizando el elemento "SegmentList" en el cual se definen desde un principio todas las URLs de los segmentos que se van a utilizar. Por tanto, es un método sencillo de aplicar para vídeo bajo demanda puesto que conocemos el número total de segmentos. Sin embargo, para la reproducción en directo existe otro mecanismo de generación de URLs más propicio mediante el elemento "SegmentTemplate".

La característica de este elemento es el uso de una plantilla para la creación de URLs. El funcionamiento consiste en sustituir una serie de variables introducidas en la plantilla cuyo valor cambia dinámicamente. En nuestro caso sólo haremos uso de la variable *$Number$,* la cual se trata simplemente de un contador de segmentos. Por tanto, esta variable tomará el valor correspondiente a la posición ocupada por el próximo segmento a solicitar en la presentación multimedia.

Por otro lado, dentro del elemento SegmentTemplate debemos destacar varios atributos :

- **@duration :** Duración del segmento. El uso de este parámetro implica que todos los segmentos direccionados por la plantilla van a tener la misma duración.

- **@initialization :** Especifica la URL correspondiente al segmento de inicialización del vídeo. Este segmento es requerido por el usuario para al unirse al streaming poder iniciar la reproducción.

- **@media :** Especifica la plantilla mediante la cual se generan todos los segmentos de vídeo de una Representation determinada. Si nos basamos en nuestro ejemplo, en el que se hace uso de la variable *$Number$* el resultado sería:

  media="dash_1_bbbh500k_$Number$.m4s"

Así pues en el ejemplo que acabamos de mostrar los valores de las URLs generadas serán -> *dash_1_bbbh500k_1.m4s, dash_1_bbbh500k_2.m4s, dash_1_bbbh500k_3.m4s ...*
Con respecto a la disponibilidad de acceso a cada uno de ellos para un tipo de presentación "dynamic", es decir en directo, el intervalo de tiempo vendrá dado por la suma de una serie de parámetros. Para un segmento "k" el cálculo sería el siguiente:

- **Segment "k" Availability Start Time :** *MPD@availabilityStartTime + PeriodStart del periodo correspondiente + Start time segmento k*

Para el obtener el valor del Start time se calcularía de la siguiente forma :

*Start time segmento k = @duration * (k -1)*

- **Segment "k" Availability End Time :** *MPD@availabilityStartTime + PeriodStart del periodo correspondiente + Start time segmento k + @duration*

4.3.2 Generación del fichero MPD

Dado que se trata de un streaming de vídeo en directo, nos vemos en la necesidad de enviar varios ficheros MPDs a los largo de la emisión. En otras palabras, al estar generando al vuelo los contenidos de la representación multimedia, se irán notificando las correspondientes actualizaciones a los usuarios cada un cierto periodo de tiempo. Por este motivo se introduce en el servidor un nuevo script llamado "timeMPD".

La funcionalidad del programa es la de actualizar el fichero MPD que le indiquemos sustituyendo principalmente el valor del parámetro *MPD@publishTime* por la hora actual. De este modo, tal como se explicó en el capítulo anterior 4.3.1.1, al enviarle el nuevo fichero MPD al usuario se detectará como actualización del streaming actual al poseer un valor *@publishTime* más reciente que el anterior. Para ello este pequeño script será utilizado cuando sea necesario por el main thread que representa el script "liveFLUTE" en el servidor de contenidos.

Cabe mencionar también otro aspecto relacionado con el funcionamiento de "timeMPD" con respecto al valor utilizado para *MPD@availabilityStartTime*. Este valor se toma en el primer fichero MPD del streaming ya que indica la fecha exacta en la que se inicia la emisión. En nuestro caso, aprovechando las características del directo hemos introducido un pequeño truco que consiste en añadir cuatro segundos extra a la fecha tomada en el momento de la ejecución de liveFLUTE. Es decir, al iniciar el streaming establecemos que el primer segmento estará disponible en el servidor transcurridos cuatro segundos. El motivo de esta modificación es tener una mayor certeza de que los clientes puedan acceder al primer segmento a tiempo ya que la red introduce un retardo. Como ya hemos mencionado. esto es posible gracias a la generación de los segmentos en tiempo real, puesto que primero el servidor genera y envía el segmento init y posteriormente tras grabar durante dos segundos, que es la duración del segmento de vídeo, envía el primer segmento. Si sumamos un cierto retardo de propagación por la red, obtenemos el valor aproximado de 4 segundos.

5. EXPERIMENTOS LIVE

En los anteriores capítulos 3 y 4 se han presentado tanto las herramientas disponibles como las que se han desarrollado con el fin de obtener un conjunto de elementos coordinados cuyo funcionamiento se asemeje al modelo propuesto por el 3GPP para la tecnología móvil LTE. Como ya se ha señalado en múltiples ocasiones a lo largo de este documento, el objetivo del trabajo es hacer uso de esta arquitectura recomendada para realizar streaming multimedia en un escenario de emisión en directo. Ahora que se conocen todos los integrantes de nuestra maqueta, es posible presentar con claridad el esquema resultante tanto en el Server como en cada Client.

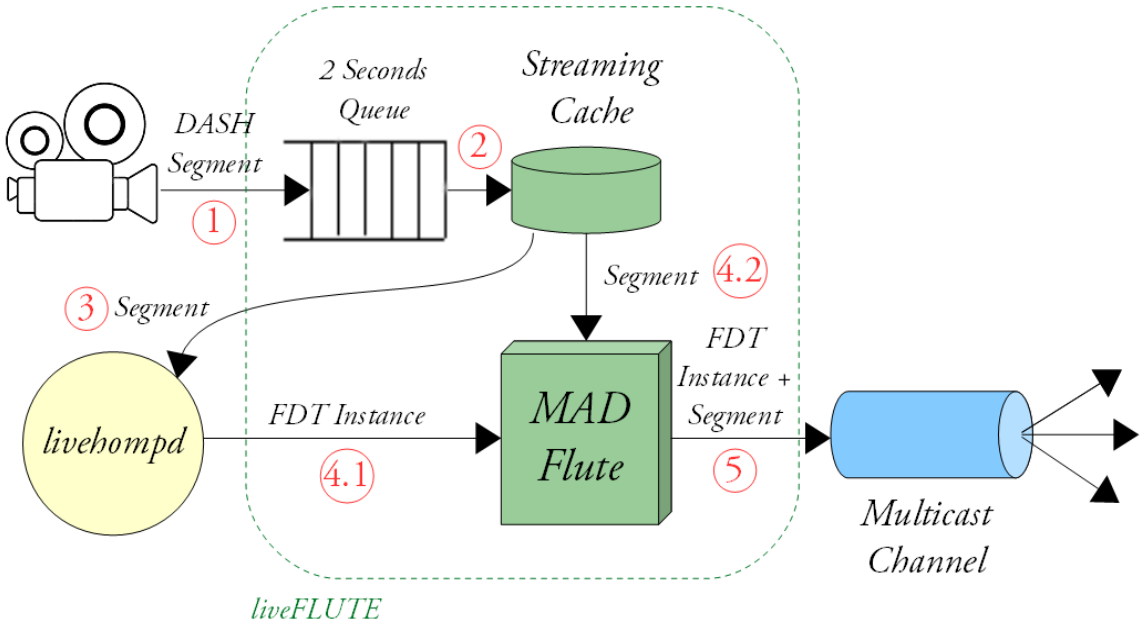

Figura 5.1: Esquema completo de flujos en el Server

En la figura 5.1 se presentan los mecanismos seguidos en el interior del servidor de distribución de contenidos multimedia para ofrecer el servicio de streaming en directo. En primer lugar, inicia la cadena de trabajo una cámara de vídeo que graba en directo el evento de interés. Acto seguido, este flujo multimedia ha de ser codificado y troceado en segmentos DASH listos para ser manejados por el Server. No obstante, se debe apuntar que esto se trataría de un escenario real, en el que en tiempo real la cámara graba y el resultado es enviado al servidor. En nuestro escenario de trabajo simplificamos esta operación partiendo con un vídeo ya codificado y troceado en segmentos DASH con una duración de dos segundos, así pues, en el esquema superior la imagen de la camára es meramente representativa para introducir el contexto ya que en la maqueta desarrollada no existe tal elemento. Para simular este comportamiento de grabación en directo encolamos dichos segmentos de vídeo y cada dos segundos se añade uno nuevo a un directorio que actúa como fuente información para FLUTE. A efectos prácticos, se consideraría como una especie de caché utilizada para el

streaming que recibe un nuevo segmento de vídeo, listo para enviar, cada dos segundos de grabación.

Una vez que el Server dispone de un segmento nuevo en esta "Streaming caché", es enviado al script "livehompd" para ser procesado por FLUTE. Recordando su definición en el capítulo 4.2.1.1, livehompd lee este segmento de vídeo y obtiene información relevante como son el path para localizarlo y su tamaño. De este modo es capaz de generar una instancia FDT que contiene los metadatos del segmento a enviar. Posteriormente, tanto la recién construida instancia FDT como su correspondiente segmento son recogidos por la implementación MAD-Flute y finalmente enviados a la red a través del canal multicast a todos los usuarios que se encuentren unidos al grupo.

Por otro lado, presentamos la segunda parte que completa todo este conjunto de engranajes que comprenden el servicio de streaming, parte correspondiente al funcionamiento en el Client. Al igual que en el Server, mostramos en la siguiente figura 5.2 el esquema final con los movimientos de todos los flujos necesarios en el Client para poder disfrutar del servicio.

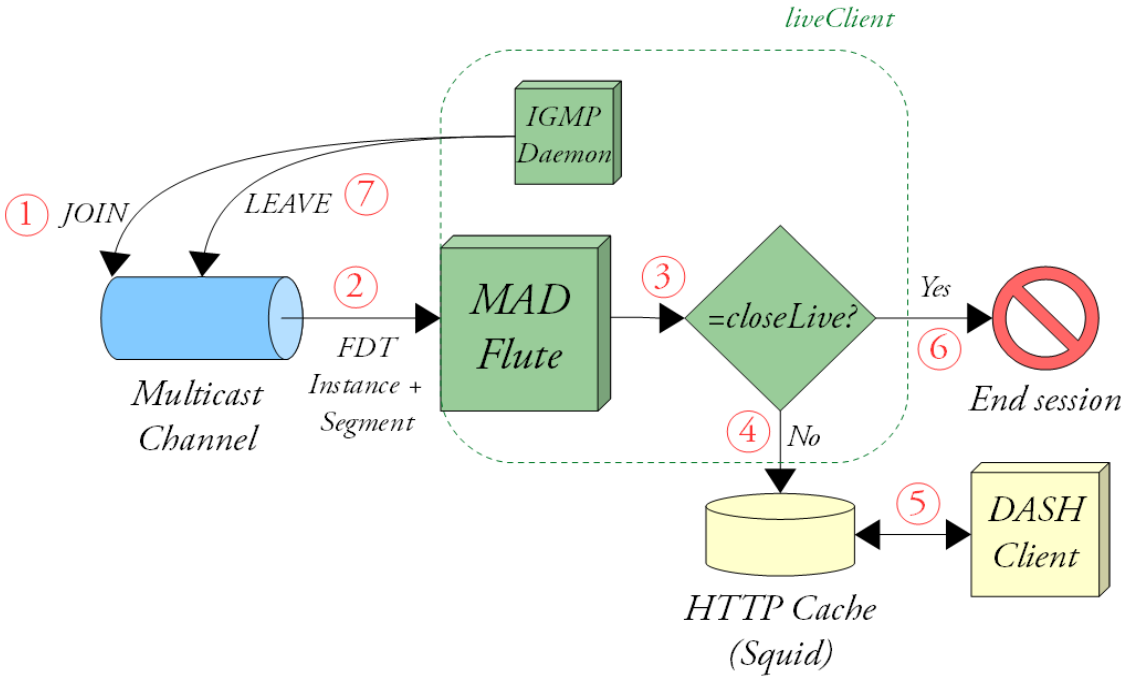

Figura 5.2: Esquema completo de flujos en el Client

En primer lugar, el usuario interesado en el streaming de vídeo ha de unirse al canal de multidifusión para poder recibir los segmentos. Para ello, internamente en el Client se ejecuta el demonio "smcroute", capítulo 4.1.3, que permite realizar peticiones IGMP. Así pues, el primer paso es enviar una petición Join al al grupo. Continuando el proceso del Server mostrado en la figura 5.1, el par compuesto por la instancia FDT y su correspondiente

segmento de vídeo llegan ahora al cliente al haberse unido al canal multicast.

Posteriormente, la instancia FDT y el segmento son recibidos por la aplicación cliente de la implementación MAD-Flute. A partir de este punto hay dos posibles caminos a seguir dependiendo de si el objeto recibido se trata del fichero "closeLive", el cual indica al Client que se terminó el streaming. Con respecto al camino habitual, representado por el paso número 4, se detecta que se trata de un segmento de vídeo nuevo, así pues es enviado a la caché HTTP. Como ya se ha explicado en detalle en el capítulo 2.2, a partir de este punto el cliente DASH se encargará de solicitar a la caché HTTP cada segmento de vídeo para visualizarlo cuando lo considere oportuno, y en caso de no encontrarlo entenderá que se ha perdido y pedirá al Server una retransmisión por unicast HTTP. Se ha de señalar que tal como se puede apreciar en la figura 5.2, no se muestra este último mecanismo ya que nos alejaríamos el ámbito que concierne al script "liveClient".

Por último, cuando el Server ya no dispone de más segmentos de vídeo nuevos significa que el streaming del evento en directo ha concluido. Para informar a todos los usuarios del servicio sobre esta novedad se envía un fichero llamado "closeLive". Dicho fichero, tal como se muestra en el paso 6.1, es leído por el cliente, el cual entiende que la sesión ha finalizado, por lo que deja de escuchar en el canal de multidifusión tras realizar una petición Leave al grupo.

5.1 Transmisión de un vídeo

Una vez asentadas las bases explicadas en la introducción previa, podemos comenzar con la primera prueba de un streaming en directo. Para ello utilizaremos el vídeo "Big Buck Bunny" en una única calidad, y utilizando la arquitectura que hemos construido procederemos al intento de emitir dicho vídeo como se si tratase de un evento en directo con el objetivo de analizar los resultados finales obtenidos. Para la ejecución de las pruebas se ha seguido el siguiente procedimiento :

1. Inicio con Google Chrome de la web con el cliente Dash.js
2. Activación del script liveClient
3. Activación del script liveFLUTE
4. Búsqueda del vídeo Big Buck Bunny en el listado
5. Inicio de la reproducción

Para depurar el funcionamiento del experimento, además de apoyarnos en los logs en tiempo real proporcionados por la caché HTTP de Squid, se accederá a la terminal proporcionada por Google Chrome donde se podrán observar los procesos ejecutados por el cliente DASH en JavaScript y de esta forma detectar con mayor facilidad errores en el transcurso de la presentación multimedia.

MPD

El elemento que adquiere una mayor importancia, y que será modificado para los diversos experimentos, se trata del fichero MPD donde definimos la información correspondiente al vídeo, para que sea correctamente interpretado por el cliente Dash.js. A continuación se presenta el contenido del fichero MPD utilizado para este experimento en concreto.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" type="dynamic" maxSegmentDuration="PT2S"
minimumUpdatePeriod="PT1S" availabilityStartTime="XXXTYYY" publishTime="XXXTYYY"
minBufferTime="PT2S" profiles="urn:mpeg:dash:profile:isoff-live:2011">
    <BaseURL>http://server.ltextreme.com/</BaseURL>
    <Period id="1" start="PT0S">
        <AdaptationSet segmentAlignment="true" maxWidth="1280" maxHeight="720"
         maxFrameRate="24" par="16:9">
            <BaseURL>streaming/</BaseURL>
            <Representation id="1" mimeType="video/mp4" codecs="avc1.4d401f" width="1280"
             height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="497236">
                <SegmentTemplate timescale="1000" duration="2000"  startNumber="1"
                 media="dash_bbbh500k_$Number$.m4s">
                    <Initialization sourceURL="dash_bbbh500k_init.mp4"/>
                </SegmentTemplate>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

Uno de los principales aspectos que se han de mencionar es la duración de la presentación multimedia. En nuestro escenario de pruebas poseemos un vídeo del cual sabemos que su duración son exactamente 10 minutos = 300 segmentos * 2 segundos. Sin embargo, en una situación en directo de verdad este valor o bien se desconoce o bien se tiene una estimación pero no exacta, por lo que corremos el riesgo de quedarnos cortos y que se detenga la emisión del evento antes de que acabe, lo que significaría una mala calidad del servicio que ofrecemos. De acuerdo al estándar de DASH, no es necesario indicar la duración total de la presentación multimedia con el parámetro *MPD@mediaPresentationDuration*, ni con la duración del periodo con *Period@duration*, nos bastaría actualizar cada cierto tiempo con una duración total desconocida haciendo uso de *MPD@minimumUpdatePeriod*. No obstante, para el experimento el valor asignado a este último experimento es de poca relevancia dado que no sufrirán ninguna actualización los contenidos transmitidos en el streaming, a la par que tampoco nos interesa manejar un intervalo de tiempo muy reducido ya que inundaríamos la red con peticiones innecesarias al Server.

Por otro lado, se pueden apreciar en el código del MPD características propias que hemos ido detallando a lo largo de este documento como es el caso de un único tipo de Representation. Al tratarse de una difusión de vídeo multicast todos los usuarios recibirán el mismo contenido multimedia, es decir, con la misma codificación, por lo que únicamente se requiere una sola Representation de 720 para nuestro experimento. También observamos como se ha utilizado el elemento XML SegmentTemplate para generar las URLs de los segmentos de vídeo. Se ha elegido ya que su funcionamiento es idóneo para streaming en directo, puesto que desconocemos el número de segmentos que utilizaremos de antemano.

Posee otro punto a favor que es la comodidad para construir el fichero MPD, ya que si se trata de un vídeo con una duración considerable tendremos que recurrir a una enorme cantidad de segmentos, que si no fuese por la plantilla tendrían que escribirse explicitamente todas sus URLs mediante el elemento SegmentList. Además, no sólo añade comodidad si no que también proporciona una mayor seguridad en la construcción del fichero porque al escribir tal cantidad de direcciones estamos incrementando el riesgo de cometer errores de modo que el cliente Dash.js falle al detectar que es un MPD erróneo.

PROBLEMAS ASOCIADOS

El principal problema que ha dificultado en gran medida conseguir una visualización del vídeo deseada era provocado por la propia implementación de Dash.js. El cliente DASH debería solicitar siempre los segmentos de vídeo a la caché HTTP en vez de solicitárselo al Server, de modo que cada segmento debe llegar al cliente justo a tiempo para ser solicitado al poco tiempo. Además ha de recalcarse el "solicitar siempre a la caché HTTP", puesto que en nuestra red no hay pérdidas, es imposible que se pierdan segmentos por el camino. El problema que se ha detectado en este experimento es debido al algoritmo con el que trabaja esta implementación de cliente DASH, ya que se aceleraba la solicitud de segmentos a la caché HTTP, en vez de solicitarlos aproximadamente cada dos segundos, que es lo que dura cada uno. Probablemente la explicación a este fenómeno sea la disposición de un enorme ancho de banda, característica debida principalmente a la virtualización de la red sin limitaciones de retardo o pérdidas, y también a la transmisión de FLUTE a una considerable velocidad de 2,5 Mbps, valor estimado que se obtendría en un canal radio en LTE en condiciones muy favorables [25]. Afortunadamente, este problema se solventó con la última versión 1.4 de Dash.js

Otro obstáculo al que nos enfrentamos en este trabajo está estrechamente relacionado con el uso del elemento XML SegmentList o SegmentTemplate. El error en cuestión se cometió en los inicios del experimento puesto que en un principio se hizo uso de SegmentList, y al tratarse de un total de 300 segmentos de vídeos se detectó que el cliente DASH no conseguía reproducirlo correctamente al estar la URL de algunos segmentos mal escrita. Por lo tanto, reafirmando lo detallado en el apartado anterior con respecto a la utilización, comprobamos

que SegmenteList nos aporta un mecanismo que sería muy poco escalable dado que estamos hablando de un fragmento de vídeo de 10 minutos, una duración mucho menor a la que se vería una situación real de streaming en directo. En cambio la plantilla proporcionada por el elemento SegmentTemplate nos aporta esta capacidad de escalar fácilmente sin cometer errores excepto que hemos de asegurarnos que numeración de los segmentos es la correcta.

Por último, existe un aspecto que daña gravemente la calidad de servicio proporcionada y que aparece al unirse un segundo cliente en medio del streaming. El problema que se presenta es la inexistente sincronización entre los clientes DASH de ambos usuarios puesto que en todas las pruebas realizadas la visualización del vídeo en el cliente 2 iba adelantada a la del cliente 1. Esta desincronización es terrible para un escenario en directo, ya que por ejemplo en un partido de fútbol se podría dar la situación que nuestro vecino cante un gol antes de que nosotros lo veamos en nuestro dispositivo móvil. Solventar este obstáculo se ha declarado como extremadamente difícil y una gran cantidad de tiempo necesaria para ello, por lo que debería ser objeto de análisis en futuros estudios relacionados.

RESULTADOS OBTENIDOS

Tras hacer frente a los problemas que se acaban de detallar, se han podido obtener una serie de resultados que nos permiten valorar este primer experimento. Desde un principio el objetivo que se ha planteado para esta prueba era conseguir una visualización del vídeo, con una transmisión en directo simulada, percibiendo una calidad de servicio óptima. En este trabajo se entiende como calidad óptima un streaming del vídeo en el cual no se produzca ninguna interrupción, con el menor retardo posible con respecto al evento en tiempo real y siempre consiguiendo que todos los segmentos solicitados por DASH lleguen a tiempo a la caché HTTP a través de FLUTE. Este último aspecto requirió bastante tiempo de investigación hasta que finalmente se consiguió con la última versión que el cliente dash.js accediese a cada segmento de vídeo con un periodo constante cercano a los dos segundos, de modo que no terminase por adelantar al ritmo del flujo entrante mediante FLUTE, con el cual recordemos son aproximadamente 2 segundos la llegada de un nuevo segmento desde el Server. De este modo finalmente se ha conseguido obtener la visualización completa del vídeo sin interrupciones y sin retransmisiones unicast, es decir, siempre accediendo al segmento correspondiente desde la caché.

5.2 Transmisión infinita

Pongámonos ahora en el papel del ofertante del servicio de streaming en directo, por ejemplo, un canal de televisión. Nuestros contenidos no se van a limitar unicamente a un vídeo de 10 minutos, en realidad podría tratarse de una programación con duración ilimitada durante 24/7. Con el objetivo de testear el rendimiento en un escenario de este tipo, simplificamos esta tarea en la que se combinan una gran cantidad de vídeos emitiendo un

úncio vídeo en bucle sin final. Para ello tomamos el vídeo del primer experimento Big Buck Bunny y se intentará emitirlo en el streaming al menos una vez de forma repetida. En otras palabras, manejaremos dos periodos a lo largo de la representación multimedia cuyos contenidos son los segmentos del mismo vídeo pero unos nombrados con un 1 y otros nombrados con un 2 en la URLs para simular que se tratan de vídeos completamente distintos.

LiveFLUTE

Para lograr este tipo de transmisión se ha realizado un pequeño cambio en el script "liveFLUTE" del Server con el objetivo de introducirlo en un bucle infinito. De este modo MAD-FLUTE al concluir la transmisión de los 300 segmentos de vídeo se reinicia y comienza el ciclo de nuevo con los mismos segmentos nombrados de otra forma. Esta característica es lo que hace interesante a la modificación del script, ya que se utilizan los mismos segmentos base pero justo antes de ser enviados a la caché Streaming del Server son renombrados introduciendo en su URL el número correspondiente a la iteración del bucle en ese momento.

MPD

Como primera aproximación a nuestro objetivo, a diferencia del experimento con un vídeo, observamos en el siguiente código del fichero MPD para este escenario que se hace uso de dos Periods desde un principio. Son prácticamente idénticos salvo por el cambio en la plantilla del SegmentTemplate en cada uno, referenciando a la iteración que corresponda en el bucle.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" type="dynamic" maxSegmentDuration="PT2S"
minimumUpdatePeriod="PT598S" availabilityStartTime="XXXTYYY" publishTime="XXXTYYY"
minBufferTime="PT2S" profiles="urn:mpeg:dash:profile:isoff-live:2011">
    <BaseURL>http://server.ltextreme.com/</BaseURL>
    <Period id="1" start="PT0.00S" duration="PT600.00S">
        <AdaptationSet contentType="video" mimeType="video/mp4" segmentAlignment="true"
          maxWidth="1280" maxHeight="720" maxFrameRate="24" par="16:9">
            <BaseURL>streaming/</BaseURL>
            <SegmentTemplate timescale="1000" duration="2000"
              initialization="dash_1_bbbh500k_init.mp4" media="dash_1_bbbh500k_$Number$.m4s"/>
             <Representation id="1" mimeType="video/mp4" codecs="avc1.4d401f"
              scanType="progressive" startWithSAP="1" bandwidth="497236"/>
        </AdaptationSet>
    </Period>
    <Period id="2" duration="PT600.00S">
        <AdaptationSet contentType="video" mimeType="video/mp4" segmentAlignment="true"
          maxWidth="1280" maxHeight="720" maxFrameRate="24" par="16:9">
            <BaseURL>streaming/</BaseURL>
```

```
                <SegmentTemplate timescale="1000" duration="2000"
                  initialization="dash_2_bbbh500k_init.mp4" media="dash_2_bbbh500k_$Number$.m4s"/>
                 <Representation id="1" mimeType="video/mp4" codecs="avc1.4d401f"
                  scanType="progressive" startWithSAP="1" bandwidth="497236"/>
            </AdaptationSet>
        </Period>
</MPD>
```

Utilizando esta estructura en el MPD no se van a requerir realizar actualizaciones sobre la presentación multimedia. Sin embargo, este hecho nos supone una limitación ya que estamos informando desde el principio sobre el contenido del streaming, lo cual es mentira que conozcamos al comienzo de la sesión el vídeo que transmitiremos. Por lo tanto, lo correcto es hacer uso de las actualizaciones del MPD e informar de ello a los usuarios. Para ello se hará una modificación de la estructura del fichero MPD al iniciar la siguiente iteración del bucle, de modo que el nuevo Period añadido, tome en la plantilla de su elemento SegmentTemplate el número correspondiente al contador del bucle. Además es muy importante que este proceso este sincronizado con las solicitudes del cliente DASH, ya que el cliente debe solicitar al Server el nuevo MPD después de la modificación para detectar la actualización, al ver que el parámetro MPD@publishTime es más reciente, y así saber cómo continuar con la reproducción del vídeo.

PROBLEMAS ASOCIADOS

Todos los posibles problemas relacionado a la transmisión en directo de un vídeo fueron enfrentados y solventados en el primer experimento, por este motivo, el principal aspecto nuevo que ha presentado fallos es la transición entre periodos. Para cerciorarnos de este fallo nos centramos simplemente en enviar de antemano los dos Periods en el MPD original, es decir, sin utilizar actualizaciones como debería ser. Finalmente se detectó que la implementación DASH en el cliente Dash.js, en el momento de redacción de este documento a última versión Dash.js 1.4.0, no está capacitado para manejar una presentación multimedia en directo con varios Periods.

El veredicto final es el siguiente: si se trata de un video bajo demanda, sí que es capaz de hacer una transición correcta entre dos Periods, mientras que si se trata de una transmisión en directo el cliente se bloquea al finalizar el primer Period. La única forma de seguir adelante con la reproducción es refrescar la pantalla y volver a clickar en el botón play. Esto se ha comprobado que funciona correctamente, ya que observando la terminal de Google Chrome, tras empezar de nuevo se ha visto que solicita segmentos pertenecientes ya al segundo Period.

También cabe destacar otro gran obstáculo que se ha detectado con respecto a la

implementación cliente. Podemos observar que en los Periods se indica su duración total, lo cual como ya hemos comentado, en una situación de tiempo real no se puede dar ya que desconocemos el futuro. Siguiendo el estándar de DASH, una posible solución más acertada sería no utilizar Period@duration, ya que bastaría indicar el Period@start del siguiente Period en el MPD. Esto se podría conseguir justo apurando a la nueva actualización en el MPD en el Server, por lo que para el primer Period no haría conocer su duración completa desde el principio. No obstante, el cliente Dash.js tampoco ha sido capaz de funcionar utilizando este mecanismo.

RESULTADOS OBTENIDOS

Con lo que acabamos de explicar en el apartado anterior, se podría decir que el experimento ha sido prácticamente un fracaso. Se ha detectado que la implementación del cliente DASH utilizada en la maqueta le falta mucho desarrollo todavía, en especial, en el modelo de streamings en directo. El aspecto positivo que se podría obtener de este segundo experimento, es que se ha comprobado que al menos el cliente es capaz de actualizarse al recibir un nuevo MPD con un segundo Period añadido al original, por lo que una parte de nuestro objetivo se consiguió.

5.3 Posibles implementaciones reales

En los últimos años hemos vivido una enorme expansión en el mercado de los contenidos de vídeo, como puede ser el caso del servicio de Movistar llamado Fusión TV. En España existe una gran audiencia en los deportes, especialmente en el fútbol, que es una oferta muy jugosa dados sus reducidos precios, por lo tanto, podemos asumir que este mercado que se encuentra en pleno crecimiento tiene un gran futuro por delante. Sin embargo, por el momento la oferta se limita al acceso a estos contenidos a través de nuestra TV, o dispositivos como las tablets o los Pcs, haciendo uso de la conexión por fibra óptica que llega al domicilio. Dado este escenario, se nos presenta un nuevo camino: ¿Y si quiero acceder desde cualquier lugar simplemente con mi smartphone y la tecnología LTE?

Supongamos que hay partido de fútbol del Real Madrid a las 20:30 pero un determinado usuario sale del trabajo a las 20:00 y tiene un largo viaje hasta casa en el autobús, por lo que llegaría a casa con el partido prácticamente concluido. Así pues, dadas las características de la arquitectura que han sido analizadas a lo largo de este trabajo, este usuario podría disfrutar en directo del partido de fútbol desde su teléfono móvil al que igual que si estuviese en casa. Para ello simplemente tendría que acceder a la aplicación desarrollada para móviles, y tras loguearse en el sistema, seleccionar el streaming correspondiente al partido del Real Madrid. De esta forma tan sencilla, el usuario gracias a su smartphone con su conexión mediante la tecnología LTE, podría disfrutar sin cortes y con calidad óptima de la emisión del evento en directo.

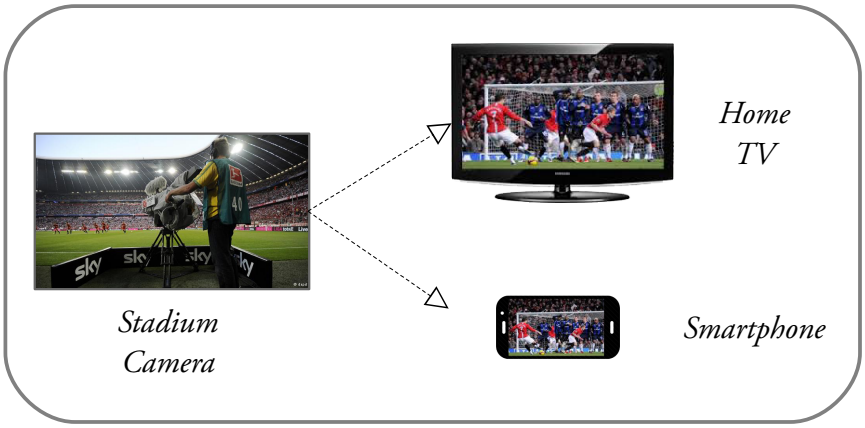

Figura 6.1: Diferentes accesos al servicio streaming

No obstante, el servicio no se limita a esta oferta, que sería la más atractiva dicho lo cual, sino que también el usuario podría acceder a toda la programación televisiva que se está emitiendo en el momento seleccionando el canal que le interese más con apenas retardo. Lo mismo se podría decir de los vídeos bajo demanda que se oferten, de modo que si le apeteciese, en vez de ver el partido de fútbol, visualizaría un capítulo de la serie Juego de Tronos o la última película de estreno en la plataforma. Además, recordemos que cada uno de estos contenidos tiene sus propias características como pueden ser los subtítulos o el idioma del audio.

En definitiva, se ve claramente que posee un amplio abanico de caracteristicas como la del servicio TV en casa, pero en este caso con la importantisima capacidad de disfrutarlo desde cualquier lugar. Así pues, se trata de una idea de negocio con una gran potencial en la cual merecería la pena invertir investigación y desarrollo.

6. CONCLUSIONES

Para finalizar con este proyecto de investigación surgido a raíz del proyecto LTExtreme, tras numerosos experimentos y un completo análisis, se puede confirmar que se ha conseguido con éxito el objetivo planteado desde un principio de transmitir un vídeo en directo siguiendo la arquitectura propuesta por el 3GPP, basándonos en la combinación del protocolo de transporte multicast FLUTE junto con el conjunto de tecnologías DASH, que hace uso de robustos mecanismos ya definidos como es el protocolo HTTP.

Se ha podido comprobar que es muy acertada la elección del protocolo FLUTE para la transmisión multicast de los segmentos, ya que se trata de un protocolo sumamente sencillo a la par que eficiente para un escenario multidifusivo. Además, dadas las características de transmisión no es necesaria establecer un sesión, o incluso un típico "handshake", para realizar

el envío de información desde el servidor de contenidos a los clientes. Por este motivo también, es muy adecuado al servicio de streaming en directo que pretendemos ofrecer, ya que el funcionamiento consiste en el envío de datos desde el servidor a la red y la posterior unión o abandono de los clientes al grupo multicast, sin que esto suponga la detención del servicio.

En cuanto a DASH, se podría decir que funciona a la perfección siempre y cuando esté correctamente sincronizado con la sesión FLUTE. Sin embargo, da la sensación de aprovecharse apenas de todo el potencial que posee esta tecnología, ya que únicamente hemos manejado una Representation, cuando la principal característica de DASH es la inteligencia para dotar al cliente de la mejor calidad posible en función del estado del enlace. Así pues, es una pena el recibir un streaming con calidad de vídeo a 480 cuando dadas las características del smartphone y un buen enlace radio podría disfrutarse a 1080p.

Este desaprovechamiento tiene su explicación a la par que nos presenta un dilema si nos ponemos en el papel del operador. En canal eMBMS está formado por múltiples slots en su interior a través de los cuales son enviados los segmentos, el problema está en la configuración de dichos slots ya que se trata de un recurso muy costoso. Así pues, se trata de un dilema en el cual tenemos la opción de configrar el canal eMBMS para transmitir por él diferentes streamings (por ejemplo, canales de TV), o bien, transmitir el mismo streaming en distintas calidades de vídeo, aprovechando más en este escenario el potencial que ofrece DASH. Por lo tanto, la elección de una de las dos opciones debe tener muchas consideraciones ya que es de gran importancia, aunque si nos ponemos en la piel del operador cabe esperar la transmisión de distintos streamings ya que sería una oferta más variada e interesante.

No obstante, debemos hablar sobre las herramientas encargadas de implementar tanto FLUTE como DASH, ya que debido especialmente a las limitaciones y los problemas que han generado, el desarrollo del trabajo se ha visto muy mermado. Con respecto a la herramienta MAD-FLUTE se ha podido comprobar que es arcaica, recordemos que la última versión data del año 2007. El procedimiento que se ha tenido que desarrollar en este proyecto para realizar las transmisiones estableciendo y cerrando una sesión por cada segmento envíado, debido a la incapacidad del programa para recibir actualizaciones de la base de datos de FDTs, es completamente ineficiente. En un escenario real, si se intentase llevar a cabo este mecanismo, tendríamos una red inundada por cientos de usuarios estableciendo sesiones constantemente con el servidor de distribución de contenidos. Además, tal como se explicó en detalle en el capítulo 4.2.1.2, el hecho de crear y cerrar sesiones para cada segmento supone el gran riesgo de no poder sincronizarse el cliente con el servidor. Por tanto, al no ser capaz de cerrar la sesión y abrir una nueva para recibir el siguiente segmento, conllevaría la pérdida de este. Por estos motivos, surgen como primera propuesta el rehacer totalmente la herramienta configurándola a nuestras necesidades y pudiendo introducir códigos más robustos como los Raptor. Mientras que otra posible propuesta sería investigar más a fondo la herramienta jFlute y buscar si es posible su integración en el escenario virtualizado, incluso recurriendo al

software Wine [22] que nos permite emular una plataforma Windows sobre la que poder ejecutar esta implementación de FLUTE en lenguaje Java.

FUTURAS MOTIVACIONES

A raíz de los resultados que se han obtenido a lo largo de este trabajo de investigación se nos presentan una serie de cuestiones y motivaciones para el futuro. En primer lugar retomamos una de las claves seguidas para implementar el modelo propuesto por el 3GPP y es el uso de una única representación de vídeo, cuando hemos detallado que DASH es capaz de manejar múltiples representaciones dependiendo de la calidad del enlace que posea el usuario. Si por ejemplo decidiésemos emitir el vídeo en directo con calidades a 720 y 1080p, una solución a primera vista sería la transmisión de dos caudales de vídeo en paralelo mediante FLUTE desde el vídeo hasta el grupo multicast gestionado por el proxy IGMP. Sin embargo, habría que analizar las capacidades típicas que posee el enlace radio ya que podríamos carecer de un ancho de banda insuficiente.

En segundo lugar, detectamos rápidamente un aspecto de gran importancia, el cual es que únicamente se ha tratado el manejo de vídeo, en cambio, el objeto es transmitir contenidos multimedia, por lo que el audio es otro componente indispensable que no hemos utilizado en los experimentos. Además, dichos contenidos poseen características extra como son los subtítulos o distintos idiomas para el audio. Por lo tanto, ¿Cómo transmitiríamos el vídeo a la par que estos contenidos? En esta ocasión, no va a ser necesario el uso de flujos paralelos al vídeo, ya que gracias al formato MP4, dentro del propio segmento pueden ir empaquetados los subtítulos y el audio en distintos idiomas, siendo el reproductor del cliente capaz de extraerlo y reproducirlo de forma sincronizada con el vídeo. Así pues, parece que el funcionamiento sería extremadamente sencillo, aunque se debería comprobar que esta característica funciona correctamente.

Como consecuencia de todos los problemas, especialmente la transición entre Periods, que ha generado la implementación Dash.js, sería muy interesante llevar a cabo estos mismos experimentos con el software BitDash, ya que parece ser un producto bastante maduro e incluso con resultados oficiales ya vistos por Internet. Así pues, la posibilidad de conseguir una gran mejora en los resultados obtenidos haciendo uso de esta implementación es bastante alta.

Por otro lado, es obvio que un potencial trabajo para el futuro sería la integración de esta implementación en un smartphone de un cliente. Para ello habría que estudiar las posibilidades de instalación en sistemas operativos como Android o iOS. Sin embargo, aquí se presenta un problema de aún mayor envergadura y es utilizar la tecnología 4G y el smartphone físicamente, dejando únicamente virtualizado el núcleo de la red y el servidor de distribución de contenidos. Cabe mencionar que ya se está trabajando en ello en el departamento del DIT y

que por el momento se ha progresado en gran medida.

Siguiendo con la integración en el cliente, una posible mejora a realizar sería obtener una automatización en la coordinación entre el script liveClient y la implementación de DASH bien mediante Dash.js, o bien, mediante el software BitDash si tras probarlo da mejores resultados que los vistos hasta el momento. En otras palabras, sería muy interesante poder conseguir que al pulsar el botón del play en el reproductor multimedia de DASH se activase inmediatamente el script liveClient para que el cliente FLUTE empiece a escuchar dicho streaming. Esta medida sería lo más cercano a la realidad, ya que de momento el proceso a seguir es manualmente ejecutar el script liveClient desde la terminal del Client y posteriormente en el reproductor DASH pulsar el botón play.

Para concluir, una muy buena futura mejora sería utilizar una cámara de vídeo y grabar en directo. Es decir, con el fin de asemejarnos a una situación real lo más posible, intentar grabar y generar segmentos de vídeo DASH al vuelo para ser enviados en directo al Client en la maqueta.

7. BIBLIOGRAFÍA

[1]. C. Cox, "An introduction to LTE: LTE, LTEAdvanced,SAE and 4G MobileCommunications", Wiley, 2012.

[2]. "Proyecto LTExtreme", http://ltextreme.org/, [Online], 2015.

[3]. "Virtual Networks over linuX (VNX)", http://dit.upm.es/vnxwiki, [Online], 2015.

[4]. ISO, "Information technology - Dynamic Adaptive Streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats," 2014, ISO/IEC 23009-1:2014.

[5]. M. Watson, M. Luby and L. Vicisano, "Forward Error Correction (FEC) Building Block", Internet RFC 5052, August 2007

[6]. T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen, "Flute — file delivery over unidirectional transport," Internet RFC 6726, November 2012.

[7]. M. Luby, M. Watson and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", Internet RFC 5775, April 2010.

[8]. M. Luby, M. Watson and L. Vicisano, "Layered Coding Transport (LCT) Building Block", Internet RFC 5651, October 2009.

[9]. L. Merritt and R. Vanam, "x264: A high performance h. 264/avc encoder," [O*nline] http://neuron2. net/library/avc/overview x264 v8 5.pdf*, 2006.

[10]. "GPAC Multimedia Open Source Project website," http://gpac.wp.mines-telecom.fr/mp4box/, [Online], 2015.

[11]. "MAD Project's Home Page", http://mad.cs.tut.fi/, [Online], 2015.

[12]. "Squid website", http://www.squid-cache.org/, [Online], 2015.

[13]. "bitdash HTML5 Player for HLS & MPEG-DASH website", http://www.dash-player.com/, [Online], 2015.

[14]. "DASH-IF website", http://dashif.org/, [Online], 2015.

[15]. "dash.js —DASH-IF Reference Client Github repository", https://github.com/Dash-Industry-Forum/dash.js.

[16]. "VideoLan organization website", http://www.videolan.org/vlc/, [Online], 2015.

[17]. J. F. Kurose, K. W. Ross, "Computer Networking: A Top-Down Approach", Adison Wesley, 6th Edition.

[18]. A. S. Tanenbaum, D. J. Wetherall, "Computer Networks", Prentice Hall, 5th Edition.

[19]. "Mcproxy - Multicast Proxy for IGMP/MLD Project website", https://mcproxy.realmv6.org/, [Online], 2015.

[20]. "Guidelines for Enterprise IP Multicast Address Allocation", http://www.cisco.com/c/dam/en/us/support/docs/ip/ip-multicast/ipmlt_wp.pdf, [Online], 2015.

[21]. "jFlute open source project", http://sourceforge.net/projects/jflute.berlios/files/, [Online], 2015.

[22]. "WineHQ website", https://www.winehq.org/, [Online], 2015.

[23]. C. Lentisco, M. Aguayo, L. Bellido, E. Pastor, D. De-Antonio-Monte, and A. García

Bolívar, “A virtualized platform for analyzing LTE broadcast services,” in European Conference on Networks and Communications (EuCNC), 2015 IEEE 24th International Conference, June 29/July 2 2015.

[24]. C. Lentisco, M. Aguayo, L. Bellido, E. Pastor,, “Supporting handover between LTE video broadcasting and unicast streaming”.

[25]. de la Fuente, A.; Lentisco, C.M.; Bellido, L.; Perez Leal, R.; Pastor, E.; Garcia Armada, A., "Analysis of the impact of FEC techniques on a multicast video streaming service over LTE," in *Networks and Communications (EuCNC), 2015 European Conference on* , vol., no., pp.219-223, June 29 2015-July 2 2015.

8. ANEXOS

8.1 Script liveClient

```
#!/bin/bash
rm logClient.txt
ENDING="closeLive"
START=""
#Sending IGMP Join Query
smcroute -j eth1 225.1.2.3
#Start listening FLUTE sessions
while [ "$START" != "closeLive" ]
do
    su -l www-data -c 'flute -A -p:4001 -t:2 -m:225.1.2.3 -B:/var/www' >> logClient.txt
    START=$(ls /var/www/ | grep $ENDING)
done
rm closeLive
echo "Live Session Ended" >> logClient.txt
#Leaving multicast group. Sending IGMP Leave Query
smcroute -l eth1 225.1.2.3
```

8.2 Script liveFLUTE

```
#!/bin/bash
COUNTER=1
mkdir streaming
TOI=1
MAX=20
DASH=2
PART=0
ACTIVE=1
FIRST=$(date +%s)
#Start infinite loop
while true;
do
#Obtaining estimated Representation duration
FOLDER="/var/www/bbbhm/"
LISTA=$( ls -l $FOLDER | wc -l)
LISTA2=$((LISTA-3)) #Susbtracting MPD and init files as they do not have duration
UPDATEPERIOD=$(( $LISTA2 * $DASH))
echo "Duration is $UPDATEPERIOD"

#Updating MPD with current date and template with its suitable transmission number
./timeMPD $COUNTER $UPDATEPERIOD
#First step: Sending init file
cp bbbhm/dash_bbbh500k_init.mp4 streaming/dash_"$COUNTER"_bbbh500k_init.mp4
sh livehompd streaming/dash_"$COUNTER"_bbbh500k_init.mp4 $TOI
flute -S -T:5 -m:225.1.2.3 -p:4001 -t:2 -r:2500 -R:/root/flute.conf -f:/root/fdt_tsi.xml
```

```
#Entering transmission loop
#Recording for 2 seconds, generating segment and giving it to FLUTE
ACTIVE=1
while [ $ACTIVE = 1 ]
do
        #Recording new video segment
        HORA=$(date +%s)
        ESPERA=$(( $HORA + $DASH))
        while [ $HORA != $ESPERA ]
        do
                HORA=$(date +%s)
        done
        #Generating new video segment
        TOI=$((TOI+1))
        PART=$((PART+1))
        #Checking remaining video segments
        NUEVO=$(ls bbbhm/ | grep dash_bbbh500k_$PART.m4s)
        if [ "$NUEVO" != "" ]
        then
                cp bbbhm/dash_bbbh500k_$PART.m4s
                        streaming/dash_"$COUNTER"_bbbh500k_$PART.m4s
                sh livehompd streaming/dash_"$COUNTER"_bbbh500k_$PART.m4s $TOI
                flute -S -T:5 -m:225.1.2.3 -p:4001 -t:2 -r:2500 -R:/root/flute.conf -f:/root/fdt_tsi.xml
        #Session is over since there are no more new segments
        else
                ACTIVE=0
                PART=0
                COUNTER=$(($COUNTER +1))
                echo "Ending transmission $COUNTER"
        fi
done
#Calculating total duration
LAST=$(date +%s)
ELAPSED=$(( $LAST - $FIRST ))
echo "Live streaming session duration $ELAPSED seconds"
#Ending infinite loop
done
#Sending close live streaming file
echo "Session ended" >> closeLive
sh livehompd closeLive $TOI
sleep 5
flute -S -T:5 -m:225.1.2.3 -p:4001 -t:2 -r:2500 -R:/root/flute.conf -f:/root/fdt_tsi.xml
echo "Sent"
```